

CONTROL TOWER FOR CLAUDE · GUIDE

What an ISO 42001 auditor may ask to see if you only use Claude

A script that builds your AI inventory from what Claude's Activity Feed shows was added, and checks it against your own register.

Published 2026-10-31 · Online version and downloads: www.datameaning.com/guides/iso-42001-ai-inventory-for-claude/

Our reading: an auditor of an ISO/IEC 42001 management system wants an inventory of the AI systems and tools you use, who owns each, and records showing the list is kept current. This guide builds that inventory from what Claude's Activity Feed shows was added, then compares it with your own AI register.

What our reading of the standard is, and what this covers

ISO/IEC 42001:2023 is a standard for an AI management system. It is written for organizations that build, provide or use AI, so a company that only uses Claude has its own part of it. The standard's text is paywalled, so we do not quote it and we name controls by id only. The table is our reading, in our words, of what each id asks a user of AI to be able to show, and the ids are mapped by us, not by ISO. It is not an official list.

Our reading, and the list that follows it

Id we map it to	Our reading	List in this guide
Clause 4.3 and A.4.2	Know which AI systems and tools are in use, and keep a list of them.	Inventory; register rows not seen
A.9	Each tool in use has an owner, an approval and a review.	Register quality
A.6.2.8	Records of events are kept, and you can show the logging stayed on.	Recording by day
A.10.3	Your suppliers are known and you hold what they can show about themselves.	Not scripted: a document you ask for

On suppliers: Anthropic says it holds a certificate for ISO/IEC 42001:2023 for its AI management system ([Anthropic: ISO 42001 announcement](#) (read 2026-10-31)). A certificate is a document Anthropic holds, and you can ask for it as supplier evidence. It is listed in Anthropic's Trust Center ([trust.anthropic.com](#) (read 2026-10-31)), where the certificate is available on request. It does not remove your own duty to know which tools your people added to Claude, which is what this guide checks.

This is evidence for your auditor. It is not a certification checklist, and it does not say any control is met.

The Compliance API covers Claude Enterprise. For Claude Console and API-key workloads, the Activity Feed records administrative and resource events but no prompt or response text. Claude Code sessions authenticated with a Console API key, run through Amazon Bedrock, Google Cloud or Microsoft Foundry, or run as Claude Code cloud sessions are not captured ([Anthropic: Compliance API FAQ](#) (read 2026-10-31)). Recording starts when the Compliance API is first turned on and is not backfilled ([Anthropic: Query the Activity Feed](#) (read 2026-10-31)). An item added before that day is not in the feed, so it needs another source.

What you will need

- The Compliance API turned on for your organization, the day you first turned it on, and a key that can read the Activity Feed: a Compliance Access Key or an Admin API key with the `read:compliance_activities` scope ([Anthropic: Query the Activity Feed](#) (read 2026-10-31)).
- Your AI register as a CSV: for each item, its id as the feed shows it, its kind, its owner, the day it was approved and the day of its next review.
- Python 3 and the `duckdb` module.

The steps

Step 1. Pull the feed from the day you turned the Compliance API on, once, as JSON lines

Each activity has an `id`, a `created_at`, an `actor` and a `type` ([Anthropic: the Activity object](#) (read 2026-10-31)). You can ask the feed for only some types with `activity_types[ ]`, but pull everything, and pull it from the day you first turned the Compliance API on, not only the period. An item added before the period and still in use has its add event before the period began; without it, the item looks unbacked, and a connector added in the missing stretch is never seen. The script checks that the pull reaches back to the day you give as `COMPLIANCE_API_ENABLED_ON` (run 3 shows what happens when it does not). Set that day: without it the pull cannot be checked, and the check fails (run 4). If an earlier period found days with no records before this period and you explained them, list those days in `KNOWN_GAP_DAYS`. The recording-by-day list needs the days with

no matching type too. Anthropic treats the feed as at-least-once and tells you to deduplicate on `id`, and to keep the starting cursor, the final `last_id`, the record count, the run time and the request-id of the final page with each pull ([Anthropic: Delivery guarantees and completeness](#) (read 2026-10-31)). The script reads a file and does not call the API. We did not call the real API, because we have no organization to call it on.

Step 2. Say which activity types add or remove an item, and load the feed and the register

Anthropic's API reference lists the types for what gets added to Claude: MCP servers and connectors (`mcp_server_created`, `mcp_server_deleted`), integrations (`claude_github_integration_created`, `claude_gdrive_integration_created`), plugins (`claude_plugin_created`, `claude_plugin_enabled`), skills (`claude_skill_created`, `claude_skill_enabled`) and desktop extensions (`desktop_extension_allowlisted`), each with a matching removal type in the table ([Anthropic: Query compliance activities, API reference](#) (read 2026-10-31)). The script holds them in a table you can read and change. Each row gives the type, a kind, the field that names the item, and whether the type adds or removes. The reference also lists `admin_connector_request_resolved` and `integration_user_connected`, which carry `mcp_server_id` too; the script does not read them; add them to the table if you want them counted.

```
ITEM_TYPES = [
    ("mcp_server_created", "mcp_server", "mcp_server_id", "add"),
    ("mcp_server_deleted", "mcp_server", "mcp_server_id", "remove"),
    ("claude_github_integration_created", "integration", "integration_id", "add"),
    ("claude_github_integration_deleted", "integration", "integration_id", "remove"),
    ("claude_gdrive_integration_created", "integration", "integration_id", "add"),
    ("claude_gdrive_integration_deleted", "integration", "integration_id", "remove"),
    ("claude_plugin_created", "plugin", "plugin_id", "add"),
    ("claude_plugin_enabled", "plugin", "plugin_id", "add"),
    ("claude_plugin_deleted", "plugin", "plugin_id", "remove"),
    ("claude_skill_created", "skill", "skill_id", "add"),
    ("claude_skill_enabled", "skill", "skill_id", "add"),
    ("claude_skill_deleted", "skill", "skill_id", "remove"),
    ("desktop_extension_allowlisted", "desktop_extension", "extension_id", "add"),
    ("desktop_extension_removed_from_allowlist", "desktop_extension", "extension_id", "remove"),
]
```

The field names in the third column are the ones the API reference shows for each type. We have not seen a real record, so open one of each type and check before you rely on it. If a record has none, the input check below counts it.

DuckDB's `read_json` takes `format='newline_delimited'` and a columns list; missing keys become NULL and keys you do not list are left out ([DuckDB: loading JSON](#) (read 2026-10-31)). The script builds that list from your table. Item ids in both files are lower-cased and trimmed, so `MCP_SRV_02` in the register and `mcp_srv_02` in the feed are the same item. The register is read as text, and a date counts only if it is written YYYY-MM-DD. DuckDB's own cast is lenient: it reads

7/1/26 as a date in the year 7. So the script checks the form first, and any other date, such as 09/24/2026 or 7/1/26, becomes NULL instead of crashing the run, and an input check counts it.

```
DATE_FORM = r"\d{4}-\d{2}-\d{2}"

def strict_date(col):
    """YYYY-MM-DD only. DuckDB's own cast is lenient: it reads 8/5/26 as the year 8."""
    c = f'trim(coalesce("{col}", \'\''))'
    return f"CASE WHEN regexp_full_match({c}, '{DATE_FORM}') THEN try_strptime({c}, '%Y-%m-%d')::DATE END"
```

Step 3. Run nine input checks before building anything

A list built from an incomplete input looks complete. The script runs nine checks first, and each one names the planted fault that trips it. The checks on records cover every record up to the period's end; a record with no readable timestamp always counts, because it cannot be placed. Faults in records after the period's end do not mark the period incomplete: they are printed on an [info] line.

- **Duplicate activity ids** (a duplicated record).
- **Records with no id or no readable created_at** (an unreadable timestamp).
- **The pull does not reach back to the day the Compliance API was turned on** (a pull that starts at the period). If COMPLIANCE_API_ENABLED_ON is not set, this cannot be checked, and the check fails instead (the enablement day not set). Quiet days right after enablement that you list in KNOWN_GAP_DAYS are skipped, so a pull that starts after them still reaches back.
- **Days with no records, from enablement (or the period's start) to its end** (five days with no records; a pull that starts at the period). When COMPLIANCE_API_ENABLED_ON falls before the period, the scan starts there, because the inventory relies on add events from before the period; otherwise it starts at the period's first day. A day before the period that you list in KNOWN_GAP_DAYS, because you explained it in an earlier period, prints on an [info] line and does not fail. Any other day before the period still fails, and a day inside the period always counts, listed or not.
- **Days in the period before the Compliance API was enabled** (the Compliance API turned on late).
- **Add or remove records that name no item** (an add record with no item id).
- **Register rows with no item id** (a register row with no item id).
- **Register items listed more than once** (a duplicated register row).
- **Register dates that are not YYYY-MM-DD** (a review date written 09/24/2026, an approval date written 7/1/26).

If any fails, all four lists are printed with `INCOMPLETE` in their heading. The script flags duplicate ids but does not remove them: remove them before you run it, keeping the first record for each id. It also prints, for information, the types in your table that never appear in the feed. A type that matches nothing means nothing was added that way, or the type is misspelled.

Step 4. Inventory: what the feed shows was added, matched to the register

The script works out each item's state from every record up to the period's end. An item is in use during the period if it was added or enabled in the period, or added earlier and not removed before the period began. Those items are joined to the register on the item id. An item in use and not on the register is `UNREGISTERED`; if its last event by the period's end is a removal it reads `UNREGISTERED (REMOVED LATER)`. For an item added before the period, the `add-events` column reads 0.

```
con.execute(f"""CREATE TABLE item_state AS SELECT kind, item,
    min(ts) FILTER (WHERE action = 'add') AS first_seen,
    count(*) FILTER (WHERE action = 'add' AND ts::DATE >= DATE '{p0}') AS adds_in_period,
    arg_max(action, ts) FILTER (WHERE ts::DATE < DATE '{p0}') AS action_before,
    arg_max(action, ts) AS last_action
    FROM events WHERE item IS NOT NULL AND ts IS NOT NULL AND ts::DATE <= DATE '{p1}' GROUP
    BY 1, 2""")
con.execute("CREATE VIEW in_use AS SELECT * FROM item_state WHERE adds_in_period > 0 OR
    action_before = 'add'")
```

```
WITH reg AS (SELECT DISTINCT item FROM register WHERE item IS NOT NULL)
SELECT u.kind, u.item, u.first_seen::DATE AS first_seen, u.adds_in_period, u.last_action,
    CASE WHEN r.item IS NOT NULL THEN 'on register'
        WHEN u.last_action = 'remove' THEN 'UNREGISTERED (REMOVED LATER)' ELSE
'UNREGISTERED' END AS status
    FROM in_use u LEFT JOIN reg r ON r.item = u.item
    ORDER BY u.first_seen, u.item"""
```

Step 5. Register rows the feed does not back up, and register quality

The second list runs the other way. A register row the feed does not show in use during the period is `ON REGISTER, NOT SEEN`; one whose last event by the period's end is a removal is `ON REGISTER, DELETED IN FEED`. Neither is proof of anything. The item may have been added before the feed started recording, or the register may be out of date, and the script asks you to find out which.

The third list gives every register row one status, the first that applies. `NO OWNER`, `NO APPROVAL DATE` and `NO REVIEW DATE` are blanks. `APPROVAL DATE UNREADABLE` and `REVIEW DATE UNREADABLE` are dates not written as a valid `YYYY-MM-DD` date, such as `7/1/26` or `09/24/2026`. `APPROVED AFTER FIRST SEEN` means the approval date is later than the first add event the feed shows for that item. `REVIEW OVERDUE` means the next review date is before `REVIEW_AS_OF`. A row with none of these reads `complete`, which says only that the fields are filled and in order.

```

CASE WHEN r.item IS NULL THEN 'NO ITEM ID'
      WHEN r.owner IS NULL THEN 'NO OWNER'
      WHEN r.approved_text = '' THEN 'NO APPROVAL DATE'
      WHEN r.approved_on IS NULL THEN 'APPROVAL DATE UNREADABLE'
      WHEN r.review_text = '' THEN 'NO REVIEW DATE'
      WHEN r.next_review IS NULL THEN 'REVIEW DATE UNREADABLE'
      WHEN f.first_seen IS NOT NULL AND r.approved_on > f.first_seen THEN 'APPROVED
AFTER FIRST SEEN'
      WHEN r.next_review < DATE '{REVIEW_AS_OF}' THEN 'REVIEW OVERDUE'
      ELSE 'complete' END AS status

```

Step 6. Recording by day, and a fingerprint on each list

The last list has one row for each day of the period with the number of records dated that day. It is the period's own record; the input check above covers the longer stretch from enablement to the period's end. A day with none that falls before COMPLIANCE_API_ENABLED_ON is BEFORE_ENABLED, because Anthropic does not backfill; any other day with none is NO_RECORDS. The logic is the one in our [HIPAA audit logs guide](#), with its cautions: a quiet day can look like an outage, and a day with some records but not all is invisible to this check. Anthropic says activity volume is not a completeness check ([Anthropic: Delivery guarantees and completeness](#) (read 2026-10-31)).

Each list is printed with its period, filters, run time, row count and a SHA-256 over every row, in the order written to its CSV:

```

def fingerprint(rows):
    body = "\n".join("|".join(" if v is None else str(v) for v in r) for r in rows)
    return hashlib.sha256(body.encode()).hexdigest()

```

The fingerprint does not prove the rows are right. It lets anyone who re-runs the script on the same files confirm they got the same rows, and, if you keep the printed fingerprint apart from the CSV, lets you check the CSV has not changed.

The code

The complete download, `ai_inventory_check.py`, also at www.datameaning.com/guides/iso-42001-ai-inventory-for-claude/ai_inventory_check.py.

```

#!/usr/bin/env python3
"""AI inventory for a company that uses Claude: what the Activity Feed shows was added, checked
against your register, in DuckDB.

python3 ai_inventory_check.py          # clean generated records
python3 ai_inventory_check.py --plant  # generated records with planted faults
python3 ai_inventory_check.py --short-pull # clean records, but pulled from the period's
start, not enablement
python3 ai_inventory_check.py --enablement-unknown # clean records, with the enablement day
not set
python3 ai_inventory_check.py --strict # exit 1 when anything needs follow-up

```

Pull the feed from the day the Compliance API was turned on, not only the period: an item added before the period and still in use has its add event before PERIOD_START.

Builds four lists, each printed with its period, filters, run time, row count and a SHA-256 of its rows:

inventory and not removed	every item in use during the period (added in the period, or added earlier before it began), matched to your AI register: on register, UNREGISTERED, or UNREGISTERED (REMOVED LATER)
register not seen REGISTER, NOT SEEN)	every register row the feed does not show in use during the period (ON or whose last event by the period's end is a removal (ON REGISTER, DELETED IN FEED)
register quality seen	every register row: owner, approval date, review date, approved before first seen
daily recording	one row per day of the period: recorded, NO RECORDS, or BEFORE ENABLED

Before it builds them it runs nine input checks and marks the lists INCOMPLETE if one fails. The record checks cover every record up to the period's end; faults in records after it are printed as [info] only. The check for days with no records runs from the day the Compliance API was turned on when that is before the period, because the inventory relies on add events from before the period; days before the period listed in KNOWN_GAP_DAYS were explained in an earlier period and print as [info]. Without COMPLIANCE_API_ENABLED_ON the pull cannot be checked, so that check fails.

The generated records are NOT a real organization's data. Field names (id, created_at, actor.type, actor.email_address, actor.user_id, type) come from Anthropic's Activity object:

<https://platform.claude.com/docs/en/manage-claude/compliance-activity-feed>

The activity-type strings come from the API reference:

<https://platform.claude.com/docs/en/api/compliance/activities/list>

The field names in the third column of ITEM_TYPES are the ones the API reference shows for each type. We have

not seen a real record, so open one of each type and check before you rely on it.

"""

```
import csv
import datetime as dt
import hashlib
import json
import pathlib
import random
import sys
```

```
import duckdb
```

```
# ---- CHANGE THESE FOR YOUR OWN DATA -----
PERIOD_START = "2026-07-06"      # first day of the period you are evidencing (UTC, inclusive)
PERIOD_END = "2026-09-27"       # last day of the period (UTC, inclusive)
COMPLIANCE_API_ENABLED_ON = None # the day you first turned the Compliance API on, YYYY-MM-DD;
None fails a check
KNOWN_GAP_DAYS = []             # days BEFORE the period with no records that you explained
earlier, YYYY-MM-DD
ACTIVITIES_FILE = None          # your Activity Feed pull, one JSON object per line; None =
generate
```

```

REGISTER_FILE = None                # your AI register (CSV with a header row); None = generate
REG_ITEM_COL = "item"              # the id of the item, as the feed shows it
REG_KIND_COL = "kind"
REG_OWNER_COL = "owner"
REG_APPROVED_COL = "approved_on"   # YYYY-MM-DD only; empty = not approved
REG_REVIEW_COL = "next_review"     # YYYY-MM-DD only
REVIEW_AS_OF = "2026-09-27"       # a review date before this day is overdue
# activity type, kind, the field that names the item (as the API reference shows it), action
ITEM_TYPES = [
    ("mcp_server_created", "mcp_server", "mcp_server_id", "add"),
    ("mcp_server_deleted", "mcp_server", "mcp_server_id", "remove"),
    ("claude_github_integration_created", "integration", "integration_id", "add"),
    ("claude_github_integration_deleted", "integration", "integration_id", "remove"),
    ("claude_gdrive_integration_created", "integration", "integration_id", "add"),
    ("claude_gdrive_integration_deleted", "integration", "integration_id", "remove"),
    ("claude_plugin_created", "plugin", "plugin_id", "add"),
    ("claude_plugin_enabled", "plugin", "plugin_id", "add"),
    ("claude_plugin_deleted", "plugin", "plugin_id", "remove"),
    ("claude_skill_created", "skill", "skill_id", "add"),
    ("claude_skill_enabled", "skill", "skill_id", "add"),
    ("claude_skill_deleted", "skill", "skill_id", "remove"),
    ("desktop_extension_allowlisted", "desktop_extension", "extension_id", "add"),
    ("desktop_extension_removed_from_allowlist", "desktop_extension", "extension_id", "remove"),
]
# -----

GEN_ENABLED = "2026-06-01"         # generated records only: the day the stand-in organization
turned the API on
OUT_DIR = pathlib.Path("evidence-out")
GEN_DIR = pathlib.Path("generated")
FIELD = {t: f for t, _, f, _ in ITEM_TYPES}

def generate(mode):
    """Seeded stand-in for an organization's feed and AI register. mode: clean, plant or short."""
    plant = mode == "plant"
    rnd = random.Random(223)
    GEN_DIR.mkdir(exist_ok=True)
    users = [f"u{n:02d}@example.test" for n in range(1, 41)]
    admin = "admin@example.test"
    start, end = dt.date.fromisoformat(GEN_ENABLED), dt.date.fromisoformat(PERIOD_END) +
dt.timedelta(days=3)
    recs = []

    def user(email):
        return {"type": "user_actor", "email_address": email, "user_id": "user_" + email.split("@")
[0],
                "ip_address": "192.0.2.1", "user_agent": "generated"}

    def add(day, typ, actor, item=None, hour=None):
        t = dt.datetime.combine(day, dt.time(hour if hour is not None else rnd.randrange(7, 19),
rnd.randrange(60), rnd.randrange(60)))
        r = {"id": "", "created_at": t.strftime("%Y-%m-%dT%H:%M:%SZ"), "organization_id":
"org_generated",
            "actor": actor, "type": typ}
        if item is not None:
            r[FIELD[typ]] = item
        recs.append(r)

```

```

day = start
while day <= end:
    add(day, "sso_login_succeeded", user(rnd.choice(users)))
    for _ in range(rnd.randrange(3, 8)):
        add(day, rnd.choice(["claude_chat_created", "claude_chat_viewed"]),
user(rnd.choice(users)))
        day += dt.timedelta(days=1)
    d = lambda m, n: dt.date(2026, m, n)
    events = [(d(6, 15), "claude_github_integration_created", admin, "gh_int_00"), # added before
the period, still in use
                (d(7, 8), "mcp_server_created", admin, "mcp_srv_01"), (d(7, 9),
"claude_github_integration_created", admin, "gh_int_01"),
                (d(7, 14), "claude_gdrive_integration_created", admin, "gd_int_01"), (d(7, 21),
"mcp_server_created", admin, "mcp_srv_02"),
                (d(7, 28), "desktop_extension_allowlisted", admin, "ext_01"), (d(8, 4),
"claude_skill_created", users[4], "skill_01"),
                (d(8, 5), "claude_skill_enabled", users[6], "skill_01"), (d(8, 18),
"mcp_server_created", admin, "mcp_srv_03"),
                (d(8, 19), "claude_plugin_created", admin, "plugin_01"), (d(8, 20),
"claude_plugin_enabled", users[11], "plugin_01"),
                (d(9, 1), "claude_skill_created", users[8], "skill_02")]
    if plant:
        events += [(d(7, 15), "mcp_server_created", admin, "mcp_srv_04"), (d(9, 2),
"mcp_server_deleted", admin, "mcp_srv_04"),
                    (d(8, 25), "mcp_server_created", admin, None)] # planted: an add
record that names no item
    if mode == "short":
        events += [(d(6, 20), "mcp_server_created", admin, "mcp_srv_00")] # planted: an
unregistered connector, added
        # before the period; the short pull below starts after it, so the feed never shows it
    for day, typ, who, item in events:
        add(day, typ, who if isinstance(who, dict) else user(who), item)
    enabled = GEN_ENABLED
    if plant:
        enabled = "2026-07-08" # planted: Compliance
API turned on two days late
    recs = [r for r in recs if r["created_at"][:10] >= enabled
            and not "2026-08-10" <= r["created_at"][:10] <= "2026-08-14"] # planted: five days
with no records
    if mode == "short":
        recs = [r for r in recs if r["created_at"][:10] >= PERIOD_START] # planted: pulled from
the period, not enablement
    recs.sort(key=lambda r: (r["created_at"], r["type"], json.dumps(r["actor"], sort_keys=True)))
    for n, r in enumerate(recs, 1):
        r["id"] = f"activity_{n:06d}"
    if plant:
        noise = [i for i, r in enumerate(recs) if r["type"] == "claude_chat_viewed" and i > 100]
        recs[noise[0]]["created_at"] = "not-a-date" # planted: an
unreadable timestamp
        recs.append(dict(recs[noise[1]])) # planted: a
duplicated record
    after = [i for i, r in enumerate(recs) if r["created_at"][:10] > PERIOD_END]
    recs.append(dict(recs[after[1]])) # a duplicated record
after the period's end:
    # not the period's fault, so it is reported as [info] and the period's checks stay clean
    feed = GEN_DIR / f"feed_{mode}.jsonl"
    feed.write_text("".join(json.dumps(r) + "\n" for r in recs))

```

```

owner = "data.platform@example.test"
reg = [{"gh_int_00", "integration", "eng.lead@example.test", "2026-06-10", "2026-12-15"},
      ["mcp_srv_01", "mcp_server", owner, "2026-07-01", "2026-12-01"], ["mcp_srv_02",
"mcp_server", owner, "2026-07-15", "2026-12-01"],
      ["mcp_srv_03", "mcp_server", owner, "2026-08-10", "2026-12-01"], ["gh_int_01",
"integration", "eng.lead@example.test", "2026-07-02", "2026-12-15"],
      ["gd_int_01", "integration", owner, "2026-07-10", "2026-12-15"], ["ext_01",
"desktop_extension", "it.ops@example.test", "2026-07-20", "2026-11-30"],
      ["skill_01", "skill", "ops.manager@example.test", "2026-08-03", "2026-12-01"],
["skill_02", "skill", "ops.manager@example.test", "2026-08-30", "2026-12-01"],
      ["plugin_01", "plugin", owner, "2026-08-15", "2026-12-01"]]

if plant:
    by = {r[0]: r for r in reg}
    reg = [r for r in reg if r[0] not in ("mcp_srv_03", "gh_int_00")] # planted: a connector
in use that nobody registered
    # (gh_int_00 goes too: with the API turned on late, its June add event is not in the
planted feed)
    by["mcp_srv_01"][3] = "7/1/26" # planted: a short
date, which loose parsers read as year 7
    by["skill_02"][2] = "" # planted: a register
entry with no owner
    by["plugin_01"][3] = "" # planted: no approval
date
    by["ext_01"][3] = "2026-08-20" # planted: approved 23
days after it was first seen
    by["gh_int_01"][4] = "2026-08-01" # planted: review date
already passed
    by["gd_int_01"][4] = "09/24/2026" # planted: a date not
written YYYY-MM-DD
    by["mcp_srv_02"][0] = "MCP_SRV_02 " # planted: same id,
other case and a space: still matches
    reg += [{"mcp_old_99", "mcp_server", owner, "2026-06-01", "2026-12-01"}, # planted:
registered, never seen
            ["mcp_srv_04", "mcp_server", owner, "2026-07-01", "2026-12-01"], # planted:
registered, deleted in the feed
            ["skill_01", "skill", "ops.manager@example.test", "2026-08-03", "2026-12-01"], #
planted: a duplicated register row
            ["", "skill", owner, "2026-08-01", "2026-12-01"]] # planted: a register
row with no item id
    with open(GEN_DIR / "register.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["item", "kind", "owner", "approved_on", "next_review"])
        w.writerows(reg)
    return str(feed), str(GEN_DIR / "register.csv"), enabled

def fingerprint(rows):
    body = "\n".join("|".join(" " if v is None else str(v) for v in r) for r in rows)
    return hashlib.sha256(body.encode()).hexdigest()

def population(con, name, control, filters, sql, header, complete):
    rows = con.execute(sql).fetchall()
    OUT_DIR.mkdir(exist_ok=True)
    with open(OUT_DIR / f"{name}.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(header)

```

```

        w.writerows(rows)
    print(f"\n== {control}: {name}" + (" " if complete else "    ** INCOMPLETE: see input checks
**"))
    print(f"period      : {PERIOD_START} to {PERIOD_END} (UTC, inclusive)")
    print(f"filters       : {filters}")
    print(f"run at          : {dt.datetime.now(dt.timezone.utc):%Y-%m-%d %H:%M:%S} UTC")
    print(f"rows           : {len(rows)}")
    print(f"sha256          : {fingerprint(rows)}")
    return rows

def ranges(days):
    """Collapse sorted dates into (first, last, count) stretches."""
    out = []
    for d in days:
        if out and (d - out[-1][1]).days == 1:
            out[-1][1] = d
            out[-1][2] += 1
        else:
            out.append([d, d, 1])
    return out

DATE_FORM = r"\d{4}-\d{2}-\d{2}"

def strict_date(col):
    """YYYY-MM-DD only. DuckDB's own cast is lenient: it reads 8/5/26 as the year 8."""
    c = f'trim(coalesce("{col}", \'\''))'
    return f"CASE WHEN regexp_full_match({c}, '{DATE_FORM}') THEN try_strptime({c}, '%Y-%m-%d')::DATE END"

def show(rows):
    for r in rows:
        print(" " + " | ".join("-" if v is None else str(v) for v in r))

def main():
    strict = "--strict" in sys.argv
    mode = "plant" if "--plant" in sys.argv else "short" if "--short-pull" in sys.argv else "clean"
    acts, reg_file, enabled = ACTIVITIES_FILE, REGISTER_FILE, COMPLIANCE_API_ENABLED_ON
    if acts is None:
        acts, reg_file, enabled = generate(mode)
        if "--enablement-unknown" in sys.argv:
            enabled = None
    # planted: the
    enablement day not set
    print(f"input      : GENERATED records (not a real organization), seed 223, mode={mode}")
    else:
        if reg_file is None:
            sys.exit("set REGISTER_FILE as well as ACTIVITIES_FILE")
        print(f"input      : " + acts)
    print(f"enabled on: " + (enabled or "not set (COMPLIANCE_API_ENABLED_ON is None)")
    con = duckdb.connect()
    id_fields = sorted(set(FIELD.values()))
    cols = ('id': 'VARCHAR', 'created_at': 'VARCHAR', 'type': 'VARCHAR', "
        "'actor': 'STRUCT(type VARCHAR, email_address VARCHAR, user_id VARCHAR)', "
        + ", ".join(f"'{f}': 'VARCHAR'" for f in id_fields))

```

```

columns = "{" + cols + "}"
con.execute(f"CREATE TABLE raw AS SELECT * FROM read_json(?, format='newline_delimited',
columns={columns})", [acts])
item_expr = "CASE type " + " ".join(f"WHEN '{t}' THEN nullif(lower(trim(\"{f}\")), '')" for t,
f in FIELD.items()) + " END"
con.execute(f"""
CREATE TABLE activities AS SELECT id, type, try_cast(replace(created_at, 'Z', '')) AS
TIMESTAMP) AS ts,
lower(trim(actor.email_address)) AS actor_email, {item_expr} AS item FROM raw"""")
vals = ", ".join(f"('{t}', '{k}', '{a}')" for t, k, _, a in ITEM_TYPES)
con.execute(f"""CREATE TABLE events AS SELECT a.id, a.ts, a.type, t.kind, t.action, a.item,
a.actor_email
FROM activities a JOIN (VALUES {vals}) t(type, kind, action) ON t.type =
a.type"""")
con.execute(f"""CREATE TABLE register AS SELECT nullif(lower(trim("{REG_ITEM_COL}")), '') AS
item,
trim(coalesce("{REG_KIND_COL}", '')) AS kind, nullif(trim(coalesce("{REG_OWNER_COL}",
'')), '') AS owner,
{strict_date(REG_APPROVED_COL)} AS approved_on, trim(coalesce("{REG_APPROVED_COL}", ''))
AS approved_text,
{strict_date(REG_REVIEW_COL)} AS next_review, trim(coalesce("{REG_REVIEW_COL}", '')) AS
review_text
FROM read_csv(?, header=true, all_varchar=true)""", [reg_file])
p0, p1 = PERIOD_START, PERIOD_END
# each item's state from every record up to the period's end: in use during the period means
added in the
# period, or added earlier and not removed before the period began
con.execute(f"""CREATE TABLE item_state AS SELECT kind, item,
min(ts) FILTER (WHERE action = 'add') AS first_seen,
count(*) FILTER (WHERE action = 'add' AND ts::DATE >= DATE '{p0}') AS adds_in_period,
arg_max(action, ts) FILTER (WHERE ts::DATE < DATE '{p0}') AS action_before,
arg_max(action, ts) AS last_action
FROM events WHERE item IS NOT NULL AND ts IS NOT NULL AND ts::DATE <= DATE '{p1}' GROUP
BY 1, 2""")
con.execute("CREATE VIEW in_use AS SELECT * FROM item_state WHERE adds_in_period > 0 OR
action_before = 'add'")
en = f"DATE '{enabled}'" if enabled else "NULL::DATE"
day_sql = f"""SELECT d::DATE AS day, count(a.id) AS records,
CASE WHEN count(a.id) > 0 THEN 'recorded'
WHEN d::DATE < {en} THEN 'BEFORE ENABLED' ELSE 'NO RECORDS' END AS
status
FROM generate_series(DATE '{p0}', DATE '{p1}', INTERVAL 1 DAY) t(d)
LEFT JOIN activities a ON a.ts::DATE = d::DATE GROUP BY 1 ORDER BY 1"""

# ---- input checks: each says which planted fault trips it. Record checks cover every record
up to the
# period's end (a record with no readable timestamp cannot be placed, so it always counts);
faults after it
# are printed as [info] only.
q = lambda s: con.execute(s).fetchone()[0]
upto = f"(ts IS NULL OR ts::DATE <= DATE '{p1}')"
later = f"ts::DATE > DATE '{p1}'"
total = q("SELECT count(*) FROM activities")
before = q(f"SELECT count(*) FROM activities WHERE ts::DATE < DATE '{p0}'")
after = q(f"SELECT count(*) FROM activities WHERE {later}")
earliest = q("SELECT min(ts)::DATE FROM activities")
days = con.execute(day_sql).fetchall()
pre = [r[0] for r in days if r[2] == "BEFORE ENABLED"]

```

```

# days with no records, from the day the Compliance API was turned on (or the period's start
when that is
# not known) to the period's end: the inventory relies on add events from before the period
scan_from = enabled if enabled and enabled < p0 else p0
off = [r[0] for r in con.execute(f""SELECT d::DATE FROM generate_series(DATE '{scan_from}',
DATE '{p1}', INTERVAL 1 DAY) t(d)
                                LEFT JOIN activities a ON a.ts::DATE = d::DATE
                                WHERE {en} IS NULL OR d::DATE >= {en} GROUP BY 1 HAVING
count(a.id) = 0 ORDER BY 1""").fetchall()]
# a day before the period that was explained in an earlier period does not fail this one; days
inside the
# period always count, listed or not
known = set(KNOWN_GAP_DAYS)
explained = [d for d in off if str(d) < p0 and str(d) in known]
off = [d for d in off if d not in explained]
# not evaluated is a failure: without the enablement day the pull cannot be checked. Quiet days
right after
# enablement that were explained earlier (KNOWN_GAP_DAYS) move the day the pull must reach back
to forward.
reach_to = enabled
if enabled:
    d = dt.date.fromisoformat(enabled)
    while str(d) in known and str(d) < p0:
        d += dt.timedelta(days=1)
    reach_to = str(d)
short_pull = 1 if not enabled or (earliest and str(earliest) > reach_to) else 0
reach_label = ("the pull does not reach back to the day the Compliance API was turned on" if
enabled else
               "the day the Compliance API was turned on is not set, so the pull cannot be
checked")
checks = [
    ("duplicate activity ids", q(f"SELECT count(*) FROM (SELECT id FROM activities WHERE {upto}
GROUP BY id HAVING count(*) > 1)"), "a duplicated record"),
    ("records with no id or no readable created_at", q(f"SELECT count(*) FROM activities WHERE
ts IS NULL OR (id IS NULL AND {upto})"), "an unreadable timestamp"),
    (reach_label, short_pull, "a pull that starts at the period; the enablement day not set"),
    ("days with no records, from enablement (or the period's start) to its end", len(off),
"five days with no records; a pull that starts at the period"),
    ("days in the period before the Compliance API was enabled", len(pre), "the Compliance API
turned on late"),
    ("add or remove records that name no item", q(f"SELECT count(*) FROM events WHERE item IS
NULL AND {upto}"), "an add record with no item id"),
    ("register rows with no item id", q("SELECT count(*) FROM register WHERE item IS NULL"), "a
register row with no item id"),
    ("register items listed more than once", q("SELECT count(*) FROM (SELECT item FROM register
WHERE item IS NOT NULL GROUP BY item HAVING count(*) > 1)"), "a duplicated register row"),
    ("register dates that are not YYYY-MM-DD",
q("SELECT count(*) FROM register WHERE (approved_text <> '' AND approved_on IS NULL) OR
(review_text <> '' AND next_review IS NULL)"),
    "a review date written 09/24/2026, an approval date written 7/1/26"),
]
print(f"\n== Input checks (records read: {total}; before the period: {before}, read for items
added earlier and their removals; "
      f"after the period's end: {after}, not used)")
ok = True
for label, n, fault in checks:
    ok &= n == 0
    print(f"[{'ok' if n == 0 else 'FAIL'}] {label}: {n}" + (" if n == 0 else f" (trips on:

```

```

{fault}}"))
    print(f"          earliest record: {earliest}; enabled on: {enabled or 'not set'}"
          + (f"; must reach back to: {reach_to} (explained days at the start skipped)" if enabled
            and reach_to != enabled else ""))
    if explained:
        print("[info] days before the period with no records, explained earlier (KNOWN_GAP_DAYS): "
              + ", ".join(str(d) for d in explained))
    inside = sorted(d for d in known if p0 <= d <= p1)
    if inside:
        print("[info] KNOWN_GAP_DAYS inside the period are not accepted, they still count: " + ",
              ".join(inside))
    for label, ds in [("days with no records", off), ("days before enablement", pre)]:
        if ds:
            print(f"          {label}: " + ", ".join(f"{a}..{b}" if a != b else str(a) for a, b, _ in
              ranges(ds)))
    late = [q(f"SELECT count(*) FROM (SELECT id FROM activities WHERE {later} GROUP BY id HAVING
count(*) > 1)"),
            q(f"SELECT count(*) FROM activities WHERE id IS NULL AND {later}"),
            q(f"SELECT count(*) FROM events WHERE item IS NULL AND {later}")]
    print(f"[info] after the period's end, not checked: duplicate ids {late[0]}, records with no id
{late[1]}, "
          f"add or remove records that name no item {late[2]}")
    seen = {r[0] for r in con.execute("SELECT DISTINCT type FROM activities").fetchall()}
    print("[info] item types never seen in this feed: " + (" ", ".join(t for t in FIELD if t not in
seen) or "none"))

    # ---- 4.3 / A.4.2: every item the feed shows added or enabled in the period, matched to the
    register
    sql = f"""
        WITH reg AS (SELECT DISTINCT item FROM register WHERE item IS NOT NULL)
        SELECT u.kind, u.item, u.first_seen::DATE AS first_seen, u.adds_in_period, u.last_action,
            CASE WHEN r.item IS NOT NULL THEN 'on register'
                WHEN u.last_action = 'remove' THEN 'UNREGISTERED (REMOVED LATER)' ELSE
'UNREGISTERED' END AS status
        FROM in_use u LEFT JOIN reg r ON r.item = u.item
        ORDER BY u.first_seen, u.item"""
    r_inv = population(con, "inventory", "4.3 / A.4.2",
                      "items in use during the period: an add or enable event in it, or an earlier
one with no removal before it began (one row per kind and item; last action as of the period's
end); matched to the register on the lower-cased, trimmed item id",
                      sql, ["kind", "item", "first_seen", "adds_in_period", "last_action",
"status"], ok)
    show(r_inv)

    # ---- register rows that the feed does not back up
    sql = f"""
        WITH used AS (SELECT DISTINCT item FROM in_use),
            last_ev AS (SELECT item, arg_max(action, ts) AS last_action FROM events
                WHERE item IS NOT NULL AND ts IS NOT NULL AND ts::DATE <= DATE '{p1}'
GROUP BY 1)
        SELECT r.item, r.kind, coalesce(r.owner, '-') AS owner,
            CASE WHEN l.last_action = 'remove' THEN 'ON REGISTER, DELETED IN FEED' ELSE 'ON
REGISTER, NOT SEEN' END AS status
        FROM register r LEFT JOIN used u ON u.item = r.item LEFT JOIN last_ev l ON l.item = r.item
        WHERE r.item IS NOT NULL AND (l.last_action = 'remove' OR u.item IS NULL) ORDER BY
r.item"""
    r_ns = population(con, "register_not_seen", "4.3 / A.4.2",
                      "register rows the feed does not show in use during the period, or whose last

```

```

event by the period's end is a removal", sql,
        ["item", "kind", "owner", "status"], ok)

show(r_ns)

# ---- A.9: is every register row owned, approved before first use, and reviewed on time?
sql = f"""
    WITH first_ev AS (SELECT item, min(first_seen)::DATE AS first_seen FROM item_state GROUP BY
1)
    SELECT coalesce(r.item, '-') AS item, r.kind, coalesce(r.owner, '-') AS owner,
    r.approved_on, r.next_review, f.first_seen,
        CASE WHEN r.item IS NULL THEN 'NO ITEM ID'
            WHEN r.owner IS NULL THEN 'NO OWNER'
            WHEN r.approved_text = '' THEN 'NO APPROVAL DATE'
            WHEN r.approved_on IS NULL THEN 'APPROVAL DATE UNREADABLE'
            WHEN r.review_text = '' THEN 'NO REVIEW DATE'
            WHEN r.next_review IS NULL THEN 'REVIEW DATE UNREADABLE'
            WHEN f.first_seen IS NOT NULL AND r.approved_on > f.first_seen THEN 'APPROVED
AFTER FIRST SEEN'
            WHEN r.next_review < DATE '{REVIEW_AS_OF}' THEN 'REVIEW OVERDUE'
            ELSE 'complete' END AS status
    FROM register r LEFT JOIN first_ev f ON f.item = r.item ORDER BY r.item NULLS FIRST,
    r.approved_on"""
    r_q = population(con, "register_quality", "A.9",
        f"every register row; first applicable status; review overdue means before
{REVIEW_AS_OF}", sql,
        ["item", "kind", "owner", "approved_on", "next_review", "first_seen",
"status"], ok)
    show(r_q)

# ---- A.6.2.8: was the feed recording on every day of the period?
r_log = population(con, "recording_daily", "A.6.2.8",
    "one row per day; records counted by created_at date; enabled on " +
(enabled or "not set"), day_sql,
    ["day", "records", "status"], ok)
print(f"  days recorded: {sum(1 for r in r_log if r[2] == 'recorded'))} of {len(r_log)}")
for st in ("NO RECORDS", "BEFORE ENABLED"):
    for a, b, n in ranges([r[0] for r in r_log if r[2] == st]):
        print(f"  {st}: {a} to {b} ({n} day(s))")
print(f"  from {scan_from} to the period's end, days with no records: {len(off)}")

follow = (sum(1 for r in r_inv if r[5] != "on register") + len(r_ns) + sum(1 for r in r_q if
r[6] != "complete")
    + len(off) + len(pre))
print("\n== Result")
print(f"input checks : {'all passed' if ok else 'FAILED, the lists above are not complete'}")
print(f"follow-up    : {follow} item(s) need an explanation "
    f"(unregistered items, register rows not seen, register rows incomplete, days with no
records or before enablement)")
verdict = "NOTHING TO FOLLOW UP" if ok and follow == 0 else "EXCEPTIONS FOUND"
print(f"RESULT: {verdict}")
return 1 if strict and verdict != "NOTHING TO FOLLOW UP" else 0

if __name__ == "__main__":
    sys.exit(main())

```

The runs

Run 1: planted faults

Command: `python3 ai_inventory_check.py --plant`

```
input      : GENERATED records (not a real organization), seed 223, mode=plant
enabled on: 2026-07-08

== Input checks (records read: 482; before the period: 0, read for items added earlier and their
removals; after the period's end: 19, not used)
[FAIL] duplicate activity ids: 1 (trips on: a duplicated record)
[FAIL] records with no id or no readable created_at: 1 (trips on: an unreadable timestamp)
[ok ] the pull does not reach back to the day the Compliance API was turned on: 0
[FAIL] days with no records, from enablement (or the period's start) to its end: 5 (trips on:
five days with no records; a pull that starts at the period)
[FAIL] days in the period before the Compliance API was enabled: 2 (trips on: the Compliance API
turned on late)
[FAIL] add or remove records that name no item: 1 (trips on: an add record with no item id)
[FAIL] register rows with no item id: 1 (trips on: a register row with no item id)
[FAIL] register items listed more than once: 1 (trips on: a duplicated register row)
[FAIL] register dates that are not YYYY-MM-DD: 2 (trips on: a review date written 09/24/2026, an
approval date written 7/1/26)
    earliest record: 2026-07-08; enabled on: 2026-07-08
    days with no records: 2026-08-10..2026-08-14
    days before enablement: 2026-07-06..2026-07-07
[info] after the period's end, not checked: duplicate ids 1, records with no id 0, add or remove
records that name no item 0
[info] item types never seen in this feed: claude_github_integration_deleted,
claude_gdrive_integration_deleted, claude_plugin_deleted, claude_skill_deleted,
desktop_extension_removed_from_allowlist

== 4.3 / A.4.2: inventory ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : items in use during the period: an add or enable event in it, or an earlier one with no
removal before it began (one row per kind and item; last action as of the period's end); matched to
the register on the lower-cased, trimmed item id
run at      : 2026-10-31 15:28:55 UTC
rows        : 10
sha256      : 3c4e452af8fcebfd006d24a79b90f2f4b0107f31e89c48a67594c197a6036c67
mcp_server | mcp_srv_01 | 2026-07-08 | 1 | add | on register
integration | gh_int_01 | 2026-07-09 | 1 | add | on register
integration | gd_int_01 | 2026-07-14 | 1 | add | on register
mcp_server | mcp_srv_04 | 2026-07-15 | 1 | remove | on register
mcp_server | mcp_srv_02 | 2026-07-21 | 1 | add | on register
desktop_extension | ext_01 | 2026-07-28 | 1 | add | on register
skill | skill_01 | 2026-08-04 | 2 | add | on register
mcp_server | mcp_srv_03 | 2026-08-18 | 1 | add | UNREGISTERED
plugin | plugin_01 | 2026-08-19 | 2 | add | on register
skill | skill_02 | 2026-09-01 | 1 | add | on register

== 4.3 / A.4.2: register_not_seen ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : register rows the feed does not show in use during the period, or whose last event by
the period's end is a removal
run at      : 2026-10-31 15:28:55 UTC
```

```

rows      : 2
sha256    : 9837749ec728c0d3a3eb13070cfaf63a59252589125795cdb4e956266aca7cbc
  mcp_old_99 | mcp_server | data.platform@example.test | ON REGISTER, NOT SEEN
  mcp_srv_04 | mcp_server | data.platform@example.test | ON REGISTER, DELETED IN FEED

== A.9: register_quality  ** INCOMPLETE: see input checks **
period    : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters   : every register row; first applicable status; review overdue means before 2026-09-27
run at    : 2026-10-31 15:28:55 UTC
rows      : 12
sha256    : f3a773cc740888cd2ac7fc1af0418a8782cd44279dcf3204ed108ff5401112b8
  - | skill | data.platform@example.test | 2026-08-01 | 2026-12-01 | - | NO ITEM ID
  ext_01 | desktop_extension | it.ops@example.test | 2026-08-20 | 2026-11-30 | 2026-07-28 |
APPROVED AFTER FIRST SEEN
  gd_int_01 | integration | data.platform@example.test | 2026-07-10 | - | 2026-07-14 | REVIEW DATE
UNREADABLE
  gh_int_01 | integration | eng.lead@example.test | 2026-07-02 | 2026-08-01 | 2026-07-09 | REVIEW
OVERDUE
  mcp_old_99 | mcp_server | data.platform@example.test | 2026-06-01 | 2026-12-01 | - | complete
  mcp_srv_01 | mcp_server | data.platform@example.test | - | 2026-12-01 | 2026-07-08 | APPROVAL
DATE UNREADABLE
  mcp_srv_02 | mcp_server | data.platform@example.test | 2026-07-15 | 2026-12-01 | 2026-07-21 |
complete
  mcp_srv_04 | mcp_server | data.platform@example.test | 2026-07-01 | 2026-12-01 | 2026-07-15 |
complete
  plugin_01 | plugin | data.platform@example.test | - | 2026-12-01 | 2026-08-19 | NO APPROVAL DATE
  skill_01 | skill | ops.manager@example.test | 2026-08-03 | 2026-12-01 | 2026-08-04 | complete
  skill_01 | skill | ops.manager@example.test | 2026-08-03 | 2026-12-01 | 2026-08-04 | complete
  skill_02 | skill | - | 2026-08-30 | 2026-12-01 | 2026-09-01 | NO OWNER

== A.6.2.8: recording_daily  ** INCOMPLETE: see input checks **
period    : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters   : one row per day; records counted by created_at date; enabled on 2026-07-08
run at    : 2026-10-31 15:28:55 UTC
rows      : 84
sha256    : a789403d14d1aa6cfe5bca300eef99458b52a3e27a64d5a7e0023670cc369cbe
  days recorded: 77 of 84
  NO RECORDS: 2026-08-10 to 2026-08-14 (5 day(s))
  BEFORE ENABLED: 2026-07-06 to 2026-07-07 (2 day(s))
  from 2026-07-06 to the period's end, days with no records: 5

== Result
input checks : FAILED, the lists above are not complete
follow-up    : 17 item(s) need an explanation (unregistered items, register rows not seen, register
rows incomplete, days with no records or before enablement)
RESULT: EXCEPTIONS FOUND

```

Run 2: clean records

Command: `python3 ai_inventory_check.py`

```

input      : GENERATED records (not a real organization), seed 223, mode=clean
enabled on : 2026-06-01

```

```

== Input checks (records read: 737; before the period: 222, read for items added earlier and
their removals; after the period's end: 19, not used)

```

```

[ok ] duplicate activity ids: 0
[ok ] records with no id or no readable created_at: 0
[ok ] the pull does not reach back to the day the Compliance API was turned on: 0
[ok ] days with no records, from enablement (or the period's start) to its end: 0
[ok ] days in the period before the Compliance API was enabled: 0
[ok ] add or remove records that name no item: 0
[ok ] register rows with no item id: 0
[ok ] register items listed more than once: 0
[ok ] register dates that are not YYYY-MM-DD: 0
      earliest record: 2026-06-01; enabled on: 2026-06-01
[info] after the period's end, not checked: duplicate ids 1, records with no id 0, add or remove
records that name no item 0
[info] item types never seen in this feed: mcp_server_deleted, claude_github_integration_deleted,
claude_gdrive_integration_deleted, claude_plugin_deleted, claude_skill_deleted,
desktop_extension_removed_from_allowlist

== 4.3 / A.4.2: inventory
period   : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters  : items in use during the period: an add or enable event in it, or an earlier one with no
removal before it began (one row per kind and item; last action as of the period's end); matched to
the register on the lower-cased, trimmed item id
run at   : 2026-10-31 15:28:56 UTC
rows     : 10
sha256   : cc27f875f755da4e018be1f6fd8b9b97ab0155a06f8129df8e0d48168400f9ba
integration | gh_int_00 | 2026-06-15 | 0 | add | on register
mcp_server  | mcp_srv_01 | 2026-07-08 | 1 | add | on register
integration | gh_int_01 | 2026-07-09 | 1 | add | on register
integration | gd_int_01 | 2026-07-14 | 1 | add | on register
mcp_server  | mcp_srv_02 | 2026-07-21 | 1 | add | on register
desktop_extension | ext_01 | 2026-07-28 | 1 | add | on register
skill       | skill_01 | 2026-08-04 | 2 | add | on register
mcp_server  | mcp_srv_03 | 2026-08-18 | 1 | add | on register
plugin      | plugin_01 | 2026-08-19 | 2 | add | on register
skill       | skill_02 | 2026-09-01 | 1 | add | on register

== 4.3 / A.4.2: register_not_seen
period   : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters  : register rows the feed does not show in use during the period, or whose last event by
the period's end is a removal
run at   : 2026-10-31 15:28:56 UTC
rows     : 0
sha256   : e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855

== A.9: register_quality
period   : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters  : every register row; first applicable status; review overdue means before 2026-09-27
run at   : 2026-10-31 15:28:56 UTC
rows     : 10
sha256   : 3334a217671d883cc9ad2857d3e7c642f6ebc8254e6afa421c0f0c228642957f
ext_01 | desktop_extension | it.ops@example.test | 2026-07-20 | 2026-11-30 | 2026-07-28 |
complete
gd_int_01 | integration | data.platform@example.test | 2026-07-10 | 2026-12-15 | 2026-07-14 |
complete
gh_int_00 | integration | eng.lead@example.test | 2026-06-10 | 2026-12-15 | 2026-06-15 | complete
gh_int_01 | integration | eng.lead@example.test | 2026-07-02 | 2026-12-15 | 2026-07-09 | complete
mcp_srv_01 | mcp_server | data.platform@example.test | 2026-07-01 | 2026-12-01 | 2026-07-08 |
complete
mcp_srv_02 | mcp_server | data.platform@example.test | 2026-07-15 | 2026-12-01 | 2026-07-21 |

```

```

complete
  mcp_srv_03 | mcp_server | data.platform@example.test | 2026-08-10 | 2026-12-01 | 2026-08-18 |
complete
  plugin_01 | plugin | data.platform@example.test | 2026-08-15 | 2026-12-01 | 2026-08-19 | complete
  skill_01 | skill | ops.manager@example.test | 2026-08-03 | 2026-12-01 | 2026-08-04 | complete
  skill_02 | skill | ops.manager@example.test | 2026-08-30 | 2026-12-01 | 2026-09-01 | complete

== A.6.2.8: recording_daily
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : one row per day; records counted by created_at date; enabled on 2026-06-01
run at      : 2026-10-31 15:28:56 UTC
rows        : 84
sha256      : eb2ca3483c5fab1cc24b37b6f7de33f862814f4a1b0c0dfb41a9d073102480f8
  days recorded: 84 of 84
  from 2026-06-01 to the period's end, days with no records: 0

== Result
input checks : all passed
follow-up    : 0 item(s) need an explanation (unregistered items, register rows not seen, register
rows incomplete, days with no records or before enablement)
RESULT: NOTHING TO FOLLOW UP

```

Run 3: a pull that starts at the period

Command: `python3 ai_inventory_check.py --short-pull`

```

input      : GENERATED records (not a real organization), seed 223, mode=short
enabled on: 2026-06-01

== Input checks (records read: 515; before the period: 0, read for items added earlier and their
removals; after the period's end: 19, not used)
[ok ] duplicate activity ids: 0
[ok ] records with no id or no readable created_at: 0
[FAIL] the pull does not reach back to the day the Compliance API was turned on: 1 (trips on: a
pull that starts at the period; the enablement day not set)
[FAIL] days with no records, from enablement (or the period's start) to its end: 35 (trips on:
five days with no records; a pull that starts at the period)
[ok ] days in the period before the Compliance API was enabled: 0
[ok ] add or remove records that name no item: 0
[ok ] register rows with no item id: 0
[ok ] register items listed more than once: 0
[ok ] register dates that are not YYYY-MM-DD: 0
  earliest record: 2026-07-06; enabled on: 2026-06-01
  days with no records: 2026-06-01..2026-07-05
[info] after the period's end, not checked: duplicate ids 1, records with no id 0, add or remove
records that name no item 0
[info] item types never seen in this feed: mcp_server_deleted, claude_github_integration_deleted,
claude_gdrive_integration_deleted, claude_plugin_deleted, claude_skill_deleted,
desktop_extension_removed_from_allowlist

== 4.3 / A.4.2: inventory ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : items in use during the period: an add or enable event in it, or an earlier one with no
removal before it began (one row per kind and item; last action as of the period's end); matched to
the register on the lower-cased, trimmed item id
run at      : 2026-10-31 15:28:56 UTC

```

```

rows      : 9
sha256    : 80786ed816dfb83b7b85427455e0d257b91a5143bf5b186c8daccd6bce43dddb
  mcp_server | mcp_srv_01 | 2026-07-08 | 1 | add | on register
  integration | gh_int_01 | 2026-07-09 | 1 | add | on register
  integration | gd_int_01 | 2026-07-14 | 1 | add | on register
  mcp_server | mcp_srv_02 | 2026-07-21 | 1 | add | on register
  desktop_extension | ext_01 | 2026-07-28 | 1 | add | on register
  skill | skill_01 | 2026-08-04 | 2 | add | on register
  mcp_server | mcp_srv_03 | 2026-08-18 | 1 | add | on register
  plugin | plugin_01 | 2026-08-19 | 2 | add | on register
  skill | skill_02 | 2026-09-01 | 1 | add | on register

== 4.3 / A.4.2: register_not_seen ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : register rows the feed does not show in use during the period, or whose last event by
the period's end is a removal
run at      : 2026-10-31 15:28:56 UTC
rows        : 1
sha256      : d5177d627452797142f2a2b230b87409eba8a10dee77a05790948dad669b10fc
  gh_int_00 | integration | eng.lead@example.test | ON REGISTER, NOT SEEN

== A.9: register_quality ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : every register row; first applicable status; review overdue means before 2026-09-27
run at      : 2026-10-31 15:28:56 UTC
rows        : 10
sha256      : a79ae3f82a4eb3771058a46f10babd90937ad762d17a46db917ec3dca15bfe64
  ext_01 | desktop_extension | it.ops@example.test | 2026-07-20 | 2026-11-30 | 2026-07-28 |
complete
  gd_int_01 | integration | data.platform@example.test | 2026-07-10 | 2026-12-15 | 2026-07-14 |
complete
  gh_int_00 | integration | eng.lead@example.test | 2026-06-10 | 2026-12-15 | - | complete
  gh_int_01 | integration | eng.lead@example.test | 2026-07-02 | 2026-12-15 | 2026-07-09 | complete
  mcp_srv_01 | mcp_server | data.platform@example.test | 2026-07-01 | 2026-12-01 | 2026-07-08 |
complete
  mcp_srv_02 | mcp_server | data.platform@example.test | 2026-07-15 | 2026-12-01 | 2026-07-21 |
complete
  mcp_srv_03 | mcp_server | data.platform@example.test | 2026-08-10 | 2026-12-01 | 2026-08-18 |
complete
  plugin_01 | plugin | data.platform@example.test | 2026-08-15 | 2026-12-01 | 2026-08-19 | complete
  skill_01 | skill | ops.manager@example.test | 2026-08-03 | 2026-12-01 | 2026-08-04 | complete
  skill_02 | skill | ops.manager@example.test | 2026-08-30 | 2026-12-01 | 2026-09-01 | complete

== A.6.2.8: recording_daily ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : one row per day; records counted by created_at date; enabled on 2026-06-01
run at      : 2026-10-31 15:28:56 UTC
rows        : 84
sha256      : eb2ca3483c5fab1cc24b37b6f7de33f862814f4a1b0c0dfb41a9d073102480f8
  days recorded: 84 of 84
  from 2026-06-01 to the period's end, days with no records: 35

== Result
input checks : FAILED, the lists above are not complete
follow-up    : 36 item(s) need an explanation (unregistered items, register rows not seen, register
rows incomplete, days with no records or before enablement)
RESULT: EXCEPTIONS FOUND

```

Run 4: the enablement day not set

Command: `python3 ai_inventory_check.py --enablement-unknown`

```
input      : GENERATED records (not a real organization), seed 223, mode=clean
enabled on: not set (COMPLIANCE_API_ENABLED_ON is None)

== Input checks (records read: 737; before the period: 222, read for items added earlier and
their removals; after the period's end: 19, not used)
[ok ] duplicate activity ids: 0
[ok ] records with no id or no readable created_at: 0
[FAIL] the day the Compliance API was turned on is not set, so the pull cannot be checked: 1
(trips on: a pull that starts at the period; the enablement day not set)
[ok ] days with no records, from enablement (or the period's start) to its end: 0
[ok ] days in the period before the Compliance API was enabled: 0
[ok ] add or remove records that name no item: 0
[ok ] register rows with no item id: 0
[ok ] register items listed more than once: 0
[ok ] register dates that are not YYYY-MM-DD: 0
      earliest record: 2026-06-01; enabled on: not set
[info] after the period's end, not checked: duplicate ids 1, records with no id 0, add or remove
records that name no item 0
[info] item types never seen in this feed: mcp_server_deleted, claude_github_integration_deleted,
claude_gdrive_integration_deleted, claude_plugin_deleted, claude_skill_deleted,
desktop_extension_removed_from_allowlist

== 4.3 / A.4.2: inventory ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters      : items in use during the period: an add or enable event in it, or an earlier one with no
removal before it began (one row per kind and item; last action as of the period's end); matched to
the register on the lower-cased, trimmed item id
run at       : 2026-10-31 15:28:56 UTC
rows         : 10
sha256       : cc27f875f755da4e018be1f6fd8b9b97ab0155a06f8129df8e0d48168400f9ba
integration | gh_int_00 | 2026-06-15 | 0 | add | on register
mcp_server  | mcp_srv_01 | 2026-07-08 | 1 | add | on register
integration | gh_int_01 | 2026-07-09 | 1 | add | on register
integration | gd_int_01 | 2026-07-14 | 1 | add | on register
mcp_server  | mcp_srv_02 | 2026-07-21 | 1 | add | on register
desktop_extension | ext_01 | 2026-07-28 | 1 | add | on register
skill       | skill_01 | 2026-08-04 | 2 | add | on register
mcp_server  | mcp_srv_03 | 2026-08-18 | 1 | add | on register
plugin      | plugin_01 | 2026-08-19 | 2 | add | on register
skill       | skill_02 | 2026-09-01 | 1 | add | on register

== 4.3 / A.4.2: register_not_seen ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters      : register rows the feed does not show in use during the period, or whose last event by
the period's end is a removal
run at       : 2026-10-31 15:28:56 UTC
rows         : 0
sha256       : e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855

== A.9: register_quality ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters      : every register row; first applicable status; review overdue means before 2026-09-27
run at       : 2026-10-31 15:28:56 UTC
```

```

rows      : 10
sha256    : 3334a217671d883cc9ad2857d3e7c642f6ebc8254e6afa421c0f0c228642957f
  ext_01 | desktop_extension | it.ops@example.test | 2026-07-20 | 2026-11-30 | 2026-07-28 |
complete
  gd_int_01 | integration | data.platform@example.test | 2026-07-10 | 2026-12-15 | 2026-07-14 |
complete
  gh_int_00 | integration | eng.lead@example.test | 2026-06-10 | 2026-12-15 | 2026-06-15 | complete
  gh_int_01 | integration | eng.lead@example.test | 2026-07-02 | 2026-12-15 | 2026-07-09 | complete
  mcp_srv_01 | mcp_server | data.platform@example.test | 2026-07-01 | 2026-12-01 | 2026-07-08 |
complete
  mcp_srv_02 | mcp_server | data.platform@example.test | 2026-07-15 | 2026-12-01 | 2026-07-21 |
complete
  mcp_srv_03 | mcp_server | data.platform@example.test | 2026-08-10 | 2026-12-01 | 2026-08-18 |
complete
  plugin_01 | plugin | data.platform@example.test | 2026-08-15 | 2026-12-01 | 2026-08-19 | complete
  skill_01 | skill | ops.manager@example.test | 2026-08-03 | 2026-12-01 | 2026-08-04 | complete
  skill_02 | skill | ops.manager@example.test | 2026-08-30 | 2026-12-01 | 2026-09-01 | complete

== A.6.2.8: recording_daily ** INCOMPLETE: see input checks **
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : one row per day; records counted by created_at date; enabled on not set
run at      : 2026-10-31 15:28:56 UTC
rows        : 84
sha256      : eb2ca3483c5fab1cc24b37b6f7de33f862814f4a1b0c0dfb41a9d073102480f8
  days recorded: 84 of 84
  from 2026-07-06 to the period's end, days with no records: 0

== Result
input checks : FAILED, the lists above are not complete
follow-up    : 0 item(s) need an explanation (unregistered items, register rows not seen, register
rows incomplete, days with no records or before enablement)
RESULT: EXCEPTIONS FOUND

```

How to read the results

Read the lists in this order. First the input checks: if one fails, every list is marked `INCOMPLETE`, and a list marked that way is a question, not evidence. Then the inventory: each `UNREGISTERED` row is an item your people added to Claude that your register does not know. Then the register rows the feed does not back up: `ON REGISTER`, `NOT SEEN` may mean the item was added before the feed began recording, or that the register is out of date. Then register quality: a row that reads `complete` has its fields filled and in order, nothing more. Last, recording by day: a `BEFORE ENABLED` day cannot be evidenced from the feed, and a `NO RECORDS` day needs an explanation.

In run 1 the script printed `RESULT: EXCEPTIONS FOUND` and 17 items needing an explanation. In run 2 it printed `RESULT: NOTHING TO FOLLOW UP`. In run 3, where the pull started at the period, it printed `RESULT: EXCEPTIONS FOUND`, because the pull did not reach back to the day the Compliance API was turned on. In run 4, with that day not set, it also printed `RESULT: EXCEPTIONS FOUND`: an unchecked pull is a failed check. Neither result says a control is met or that anyone will pass an audit.

Checklist

Done when	Where
☐ Compliance API on; the day it was first turned on is written down	Need
☐ Activity Feed pull saved from the day the Compliance API was turned on, and the reach-back check reads ok; cursor, last_id, count, run time and request-id kept with it	Step 1
☐ Each field name in ITEM_TYPES checked against one of your own records of that type	Step 2
☐ Register item ids are written as the feed shows them; dates are YYYY-MM-DD	Step 2
☐ COMPLIANCE_API_ENABLED_ON set; KNOWN_GAP_DAYS lists only days explained in writing in an earlier period	Step 1
☐ Duplicate activity ids removed before the run	Step 3
☐ All nine input checks read ok, or each failure is explained in writing	Step 3
☐ Every UNREGISTERED item is registered, removed from Claude, or explained	Step 4
☐ Every ON REGISTER, NOT SEEN and DELETED IN FEED row is explained or corrected	Step 5
☐ Every register row that is not complete has an owner, an approval date and a review date, or a reason	Step 5
☐ Every NO RECORDS and BEFORE ENABLED stretch is explained	Step 6
☐ Row counts and SHA-256 of each list kept with the CSVs	Step 6
☐ Coverage line shown to your auditor with the lists	Limits

Limits

- **This is not a client result.** The records are generated. The run shows the script finds planted faults. It says nothing about any real organization or how often such faults happen.
- **We did not call the Compliance API,** and we have not seen a real record. The field names come from the API reference, so check one record of each type.
- **The feed shows what was added to Claude, not every AI tool your company uses.**
Coverage: The Compliance API covers Claude Enterprise. For Claude Console and API-key workloads, the Activity Feed records administrative and resource events but no prompt or response text. Claude Code sessions authenticated with a Console API key, run through Amazon Bedrock, Google Cloud or Microsoft Foundry, or run as Claude Code cloud sessions are not captured ([Anthropic: Compliance API FAQ](#) (read 2026-10-31)). Recording starts when the Compliance API is first turned on.

- **The script does not read who added an item.** In our generated records, a skill or plugin enabled by a person is an add event like any other. The script does not tell an approved enablement from an unapproved one; it checks that the item is on the register.
- **Our reading of the standard is not ISO's.** The table in the first section is ours. Your auditor and your own scope decide what the standard asks of you.
- **Disabling a plugin or skill (`claude_plugin_disabled`, `claude_skill_disabled`) is not counted as a removal.** Anthropic records each for one member's account ([Anthropic: API reference](#) (read 2026-10-31)), so one member turning it off does not mean the organization stopped using it. Add those types to the table if you want it to be.
- **A register row marked `complete` is a row with its fields filled in.** It does not show anyone reviewed the tool.
- **The pull holds workforce emails,** so store it under the access controls you use for other security records. The script runs on your machine and sends nothing to Data Meaning.

Tested on

Platform

DuckDB 1.5.6, Python 3.14, run from Python with the duckdb module

Edition

Open-source DuckDB, in-memory

Date

2026-10-31

Dataset

Generated records, seed 223 (not a real organization). Field names from [Anthropic's Activity object](#) (read 2026-10-31); activity types from [Anthropic's API reference](#) (read 2026-10-31)

Code

[ai_inventory_check.py](#)

Sources

- https://duckdb.org/docs/stable/data/json/loading_json.html (read 2026-10-31)
- <https://platform.claude.com/docs/en/api/compliance/activities/list> (read 2026-10-31)
- <https://platform.claude.com/docs/en/manage-claude/compliance-activity-feed> (read 2026-10-31)
- <https://platform.claude.com/docs/en/manage-claude/compliance-faq> (read 2026-10-31)
- <https://platform.claude.com/docs/en/manage-claude/compliance-integration-patterns> (read 2026-10-31)

- <https://trust.anthropic.com> (read 2026-10-31)
- <https://www.anthropic.com/news/anthropic-achieves-iso-42001-certification-for-responsible-ai> (read 2026-10-31)

See Control Tower for Claude

Control Tower for Claude runs in your own Azure subscription.

www.datameaning.com/solutions/claude-control-tower.html