Pinned upstream baselines

Use this reference to identify the official product source revision against which each maintained connector mapping was reviewed. A baseline freezes the route and behavior audit. It does not vendor the product, add it to the AIP build, or i

SECTION	Qualification and Evidence
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/upstream-baselines.md

At AIP source revision `97be86e9efedf07ecf1783b03800f683f107fb04`, the root implementation table and all six connector constants agree on the pins below.

Use the connector constant as the source of truth

Product	Official repository	Upstream revision	Connector constant	Reference manifest shape
Cal.diy	calcom/cal.diy	`f00434927386c9ecdcbd7e6c5f82d22044a245bc`	`aip-connector-cal-diy/src/lib.rs`	81 capabilities
Hermes Agent	nousresearch/hermes-agent	`7426c09beee73bdff94d916015bac71384f6bc92`	`aip-connector-hermes-agent/src/lib.rs`	35 capabilities per endpoint
Chatwoot	chatwoot/chatwoot	`8818d276b954ac4f84cffd8915c99f40e43804ed`	`aip-connector-chatwoot/src/operations.rs`	153 capabilities
Dify	langgenius/dify	`f8d47616c15d0959f604f6a5e2e1d32d3108991b`	`aip-connector-dify/src/operations.rs`	74 capabilities in the reference app-plus-knowledge topology
CrewAI	crewAIInc/crewAI	`bfa652a7be8637562cc9b0833f75d927a64552d1`	`aip-connector-crewai/src/operations.rs`	10 capabilities per crew
Twenty	twentyhq/twenty	`96a24563674313a3071d359bfccaf33d5e130ab8`	`aip-connector-twenty/src/lib.rs`	22 capabilities per workspace

With one endpoint, crew, and workspace, the six reference manifests contain 375 capabilities. That total is a catalogue identity, not a statement that every capability was exercised against a live product.

Each connector also includes its upstream revision in manifest compatibility metadata. Treat a mismatch among the source constant, generated manifest, documentation, and retained report as a release blocker.

Keep five identities separate

An upstream commit is only one part of reproducible connector evidence.

Identity	What it answers
Upstream source revision	Which official product code defined the audited contract?
Runtime dependency version	Which package or library did an adapter load?
Provider deployment identity	Which source tree, image, configuration, and migration state was running?
AIP connector artifact	Which connector source, image, manifest, and admission package was used?
Qualification report	Which topology and assertions were observed, and when?

These identities cannot substitute for one another. A provider image may be built from a dirty tree despite carrying an upstream tag. A connector constant may still point to the correct commit while the deployed product has advanced. A historical report may name both correctly but cover an older AIP image.

CrewAI has an additional distinction: the sidecar pins the `crewai` Python package to `1.15.5` in `pyproject.toml` and locks its dependency graph in `uv.lock`. That package version is a runtime identity. The upstream commit in the table remains the source-audit identity.

The other five connectors call product HTTP surfaces and do not compile the upstream source into their Rust crates. Their provider build identity must be captured by the product campaign.

Re-audit the right contract surface

When a pin changes, review the behavior that the connector actually depends on, not the product's release notes alone.

Product	Contract surfaces to re-audit
Cal.diy	Versioned business controllers; scheduling request and response types; booking, slot, schedule, event-type, team, routing, conferencing, OAuth-client, and webhook behavior; exact raw-body signing; trigger names; removed or operator-only routes
Hermes Agent	Health and discovery; model, skill, and toolset catalogues; chat and response streaming; durable runs and events; approvals; stop and polling behavior; sessions; schedules; MCP configuration and tool filtering
Chatwoot	Account-scoped REST routes; conversations, messages, assignments, labels, contacts, and attachments; multipart rules; mutation idempotency expectations; webhook signature input, headers, event identity, and loop prevention
Dify	Application mode detection; completion, chat, workflow, task, and cancellation routes; Service and Knowledge APIs; JSON, multipart, binary, and SSE responses; human-input and paused-workflow events
CrewAI	Async kickoff; ordered stream events; cancellation and close behavior; terminal output; batch, replay, training, testing, knowledge, and memory APIs; sidecar authentication and persistent job semantics
Twenty	Core-record paths for standard and custom objects; filters, cursors, batch actions, duplicates, group-by, merge, deletion, and restore; metadata resources; workspace OpenAPI paths; webhook HMAC input, timestamp, nonce, and secret handling

For every product, also compare:

- HTTP methods, paths, status codes, redirects, and content types;
- authentication placement, scope, rotation, and tenant or workspace binding;
- required and optional request fields plus provider defaults;
- response, pagination, stream, event, and terminal-state shapes;
- retry, cancellation, idempotency, and uncertain-outcome semantics;
- webhook raw bytes, signature construction, time window, and replay identity;
- error categories, rate limits, body limits, and deprecated surfaces.

An unchanged route name does not prove an unchanged contract.

Verify an exact local checkout

Set the expected repository, revision, and clean checkout, then perform only read-only checks:

```
SH
upstream=/absolute/path/to/product-checkout
expected_remote=https://github.com/owner/repository.git
expected_revision=0123456789abcdef0123456789abcdef01234567
```

```
test "$(git -C "$upstream" remote get-url origin)" = "$expected_remote"
test "$(git -C "$upstream" rev-parse HEAD)" = "$expected_revision"
git -C "$upstream" cat-file -e "$expected_revision^{commit}"
test -z "$(git -C "$upstream" status --porcelain)"
git -C "$upstream" show -s --format='%H %T %cI %s' "$expected_revision"
```

Record the commit and tree IDs in the audit. For a provider build, also record the deterministic source archive digest and resulting image digest. A local checkout path is not part of the public contract and must not appear in the qualification claim.

If the expected commit object is unavailable, fetch it from the verified official remote in a controlled preparation step. Do not audit a nearby branch or newer checkout and describe it as the pin.

Advance a baseline deliberately

Treat any pin change as a connector compatibility change, even when native AIP messages do not change.

1. select one full commit from the official repository;
2. verify the remote, commit, tree, and clean source state;
3. diff every product-specific surface listed above against the previous pin;
4. classify additions, removals, defaults, deprecations, and semantic changes;
5. update the connector's `UPSTREAM_REVISION` constant and only the reviewed

operation mappings;

6. update request, response, event, error, schema, approval, retry,

cancellation, webhook, and exclusion tests as affected;

7. regenerate the connector manifest and verify its capability count and

upstream metadata;

8. run the complete applicable connector conformance matrix;
9. rebuild immutable connector and provider artifacts and run the required

isolated-live campaign;

10. retain the old and new pins, artifact digests, diff review, reports, and

explicit limitations in release evidence.

Do not silently move the documentation table ahead of the source constant. Do not remove an older baseline from a historical report; that report must continue to identify what it actually tested.

State compatibility narrowly

Use wording such as:

> Connector artifact `<digest>` was reviewed and qualified against provider
>
> artifact `<digest>` built from upstream commit `<full-commit>`.

Do not say `supports the latest version`, `supports all product APIs`, or `compatible with commit X and later`. Those claims require evidence beyond a single frozen mapping.

Related documentation

- Connector documentation

- Implementation status
- Conformance and qualification
- Live product E2E