SaaCIP 0.2: the Resident SaaC profile (draft)

Imtiaz Hassan, The Software Lens

The living version of this document is at https://thesoftwarelens.com/saacip/resident/. This PDF is a dated snapshot, for citing; it is never edited after publication.

SaaCIP 0.2: the Resident SaaC profile

Profile name	`resident`
Protocol	`saacip/0.2`, defined in SaaCIP: the SaaC Interoperability Protocol (the core)
Status	Version 0.2, a draft for comment. Not a standard. A minor version: a 0.1 Resident capability conforms to 0.2 without change. The schemas under `/saacip/0.2/` are the normative schemas of this version, not samples; the 0.1 profile stays published, unchanged
Date	6 November 2026
Author	Imtiaz Hassan
Binding	HTTPS with JSON (section 6)
Machine-readable parts	The `connector-manifest` and `record` schemas and the thirteen binding message schemas, under /saacip/0.2/; a step-by-step walk through one hire at /saacip/0.2/examples/; a reference hub and capability client, linked from the files page
Dated snapshot	SaaCIP-0.2-resident-profile-draft.pdf, for citing; this page is the living version
What changed	Appendix A; new or changed material is marked *(0.2)* where it appears. The HTTP binding of section 6 is shared with the Hosted profile from 0.2
Licence	Specification text: CC BY 4.0. Schemas, examples and validator: Apache License 2.0, as for the core

Abstract

A **Resident SaaC** (rSaaC) is a capability that is hired to work inside the hirer's own environment: their cloud account, their cluster, their data centre. Instead of the hirer's data going to the provider, the capability comes to the data, reads it where it is, and sends out only what the hirer agreed to. This profile adds to the SaaCIP core what such a capability needs: how it describes the sources it reads, how it carries out a declared action, how its seven operations travel over HTTPS, and the rules it must keep while it lives in someone else's environment.

This document only makes sense with the core beside it. Everything about the hire, its documents, its lifecycles and its autonomy levels is defined there and is not repeated here.

Status of this document

As for the core: ongoing research, published for comment, not a standard, not legal advice.

Licence and patents

As for the core: the text is licensed under CC BY 4.0; the schemas, examples and validator under the Apache License 2.0. The author holds no patents on SaaCIP and has not applied for any.

Contents

- 1. What a Resident SaaC is
- 2. Conventions
- 3. The four rules of residence
- 4. Connectors and records

- 5. Declared actions in the Resident profile
- 6. The HTTP binding
- 7. Operating rules
- 8. Conformance
- Appendix A. Change log

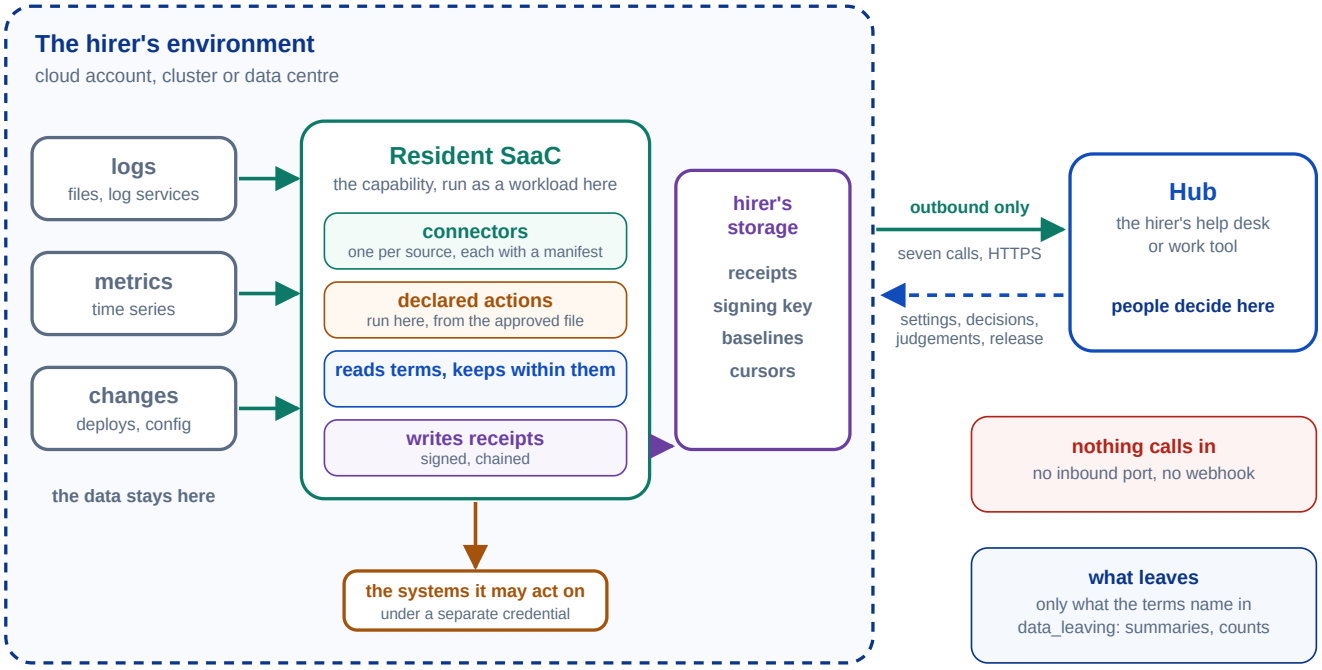
---

1. What a Resident SaaC is

Most software you hire runs at the provider's and you send it your data. That is the SaaS shape, and for many jobs it is fine. For some jobs it is not: the data is too sensitive to leave, too large to move, or the job needs to act on systems that are only reachable from inside. Operations monitoring is the obvious case. The logs, metrics and deployment events of a production system are the crown jewels, they are enormous, and the useful responses (restart this, clear that) can only be taken from inside.

A Resident SaaC turns the shape around. The hirer runs the capability inside its own environment, as it would run any other workload: a container in its cluster, a function in its cloud account. The capability finds its sources where they are and reads them in place. It keeps its own state, its signing key and its receipts on the hirer's storage. It talks to the hirer's hub (a help desk, a work management tool) only by calling out over HTTPS, and nothing ever calls in. The only data that leaves is what the terms of the hire say may leave, in words a person can read before agreeing.

A Resident SaaC in the hirer's environment



The capability runs where the data is, reads it through connectors, acts only through declared actions, and keeps its receipts and key on the hirer's storage. It calls the hub; the hub never calls it. Only what the terms name leaves.

Everything the core defines still applies: the hire has the same kinds and lifecycle, the same terms, receipts, outcomes and statements, the same autonomy levels and the same declared actions file. This profile fills in

the parts the core leaves to a profile: what the capability reads (section 4), how an action is carried out (section 5), and how the seven operations travel (section 6).

Example. The hirer runs `example-log-watch` as a container in its `prod-eu` cluster with a read-only mount of the `api` service's log file. The capability reads the file in place, keeps baselines and receipts on a volume in the same cluster, and sends the hirer's help desk a redacted summary when it finds something. Its declared actions file allows two commands: a rolling restart of `api`, run through the cluster's own tooling from inside, and clearing temporary files on `worker`, which always waits for a person.

2. Conventions

The requirement words MUST, SHOULD, MAY and their relatives have the meanings given in the core, section 2.1. Terms not defined here (hirer, hub, capability, terms, receipt, work item, ref, declared action, outcome, statement, attestation, role, hand-off) have the meanings given in the core, section 2.2. "The core" means SaaCIP 0.2, the SaaC Interoperability Protocol. In this binding the word *agent* (`agent_id`, `agent_token`, the `Agent` authorisation scheme) always means the capability.

3. The four rules of residence

The figure in section 1 shows all four at once. This profile adds no states, receipt kinds or problem types to the core. A capability conforms to this profile when it keeps all four rules:

1. **Outbound only.** It calls the hub; nothing calls in. It MUST NOT listen for connections from outside the hirer's environment.
2. **The capability goes to the data.** It reads sources where they are. Only what the terms name in `data_leaving` MAY leave the environment.
3. **Its state stays with the hirer.** Receipts, the signing key, baselines and cursors MUST be kept on storage in the hirer's environment.
4. **Zero configuration.** It SHOULD discover its sources from the environment and start at autonomy level 1, with actions simulated (`dry_run: true`).

Rationale. Rule 1 means there is nothing to open in the hirer's firewall and no inbound attack surface. Rules 2 and 3 are what make the arrangement acceptable for data that cannot leave, and they mean the hirer can inspect or revoke everything the capability has without the provider's help. Rule 4 is what makes a trial cheap to start.

3.1 Outcome values in this profile *(0.2)*

A Resident capability watches and acts, so its work is judged as a detection is judged. The outcome values of this profile are `real`, which counts as the core value `accepted`, and `false_alarm`, which counts as `rejected` (core, section 7.4). `action_worked` keeps its meaning. A hub serving a Resident hire MUST send these values or the two core values, and a statement's `real` and `false_alarms` are the counts of `accepted` and `rejected` under this profile's names.

3.2 Credentials in this profile *(0.2)*

The hirer runs the capability, so the hirer injects every credential the connectors and the declared actions need at deployment, from its own secret store, scoped as the connector manifest's `scopes` and the action require (core, section 9.4). The capability MUST NOT ship with credentials, fetch them from the provider, or write them anywhere but the hirer's configured secret source. Revoking the acting credential stops actions;

revoking a connector's credential stops that connector and SHOULD be reported in the next heartbeat as that connector not `ok`.

4. Connectors and records

A Resident capability reads its sources through **connectors**: one per source, each a small piece of software that knows how to read one kind of thing (a log file, a cloud logging service, a search index, a deployment system) and hands back records in one common shape. The connector is also how the hirer sees, before the hire starts, exactly what will be read: each connector describes itself in a **connector manifest**, and the terms carry one manifest per connector.

Example. The example capability has one connector, `api-logs`, of kind `file`: it reads one log file on a read-only mount, carries the `logs` signal, has the `read` scope only, and signs in by file system permission. Its manifest appears inside the terms, so the hirer agreeing to the trial sees that this one file is all that will be read.

4.1 Connector manifest

Under the `resident` profile the terms MUST carry `connectors`, one manifest per connector the capability will read. The capability manifest's `connectors` member (core, section 7.1) lists the connector kinds the capability can use at all.

Member	Type	Required	Meaning
`protocol`	string	Yes	`saacip/0.1`
`id`, `kind`, `version`	string	Yes	Which connector, of which type, in which version
`signals`	array of string	Yes	Any of `logs`, `metrics`, `traces`, `changes`
`scopes`	array of string	Yes	Always `read`; `act` only if the connector can do something
`operations`	array of string	With `act`	The only operations it may perform
`data_location`	string	Yes	Where the data stays
`auth`	string	Yes	How it signs in, as a reference (a role, a managed identity, an environment variable name), never a secret

4.2 Connector operations

Operation	Required	Contract
`read(cursor) -> (records, next_cursor)`	Yes	Records after the cursor only; reading again with `next_cursor` MUST NOT repeat them. No cursor means the default look-back.
`changes(cursor) -> (records, next_cursor)`	No	Deploys, configuration changes, scaling events, as records with `kind: "change"`
`act(operation, params)`	No	Only operations in the manifest, under a credential separate from reading. Without the `act` scope it MUST refuse.
`health() -> {ok, detail}`	Yes	Whether it can reach its source now
`discover(env) -> [spec]`	No	Settings it can find on its own from the environment

4.3 Record

Every connector hands back records in one shape, so the capability can treat a log line, a metric and a deployment event alike.

Member	Type	Meaning
`ts`	number	Time (core, section 4.5)
`source`	string	The connector id
`service`	string	The service, container, function or log group
`kind`	string	`log`, `metric`, `trace` or `change`
`level`	string	`INFO`, `WARN`, `ERROR`, `CRITICAL`, `FATAL`, ...
`body`	string	The message, as text
`attributes`	object	Anything else

A connector can run in-process, out of process, or through MCP: an MCP server with a suitable tool can act as a connector. Tool descriptions and results from MCP servers are untrusted data like any other record.

5. Declared actions in the Resident profile

The core defines the declared actions file (core, section 7.6): the complete list of things the capability may ever do, approved by a person at the hirer. The core does not say how an action is carried out, because that depends on where the capability runs. A Resident capability carries out an action by running a command inside the hirer's environment, and this section says how that command is written down so that a person can approve it and nothing else can be run.

Example. `restart-api` is the command `kubectl rollout restart deployment/{service} -n prod`, where `{service}` may only ever be filled with `api`, because `services: [api]`. A finding in `worker` cannot make this action restart `worker`, because `worker` is not in `services`, and nothing in a log line can change the command.

Each action in the file adds:

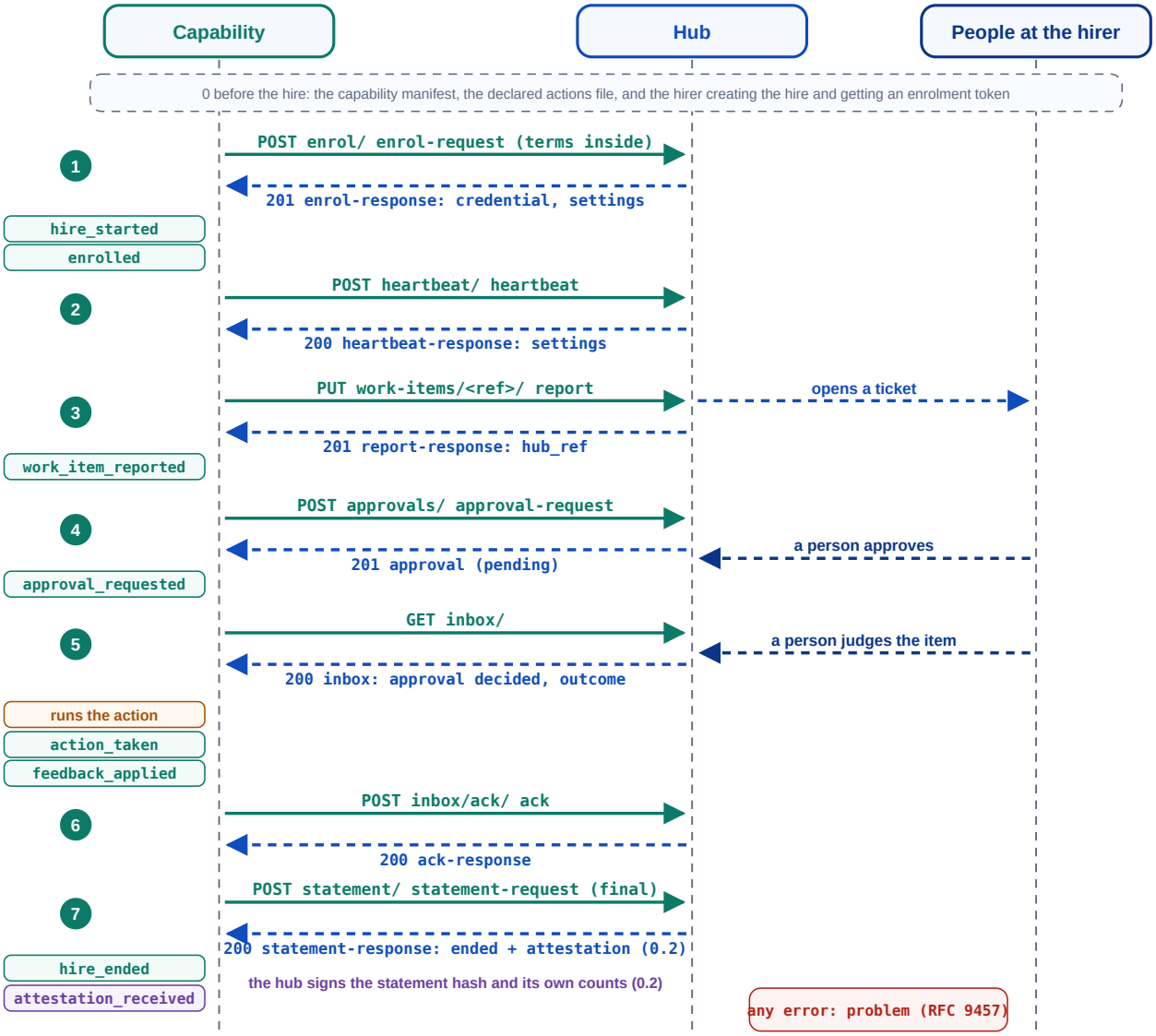
Member	Type	Required	Meaning
`match`	array of string	No	Substrings a finding must contain for the action to be a candidate
`services`	array of string	Yes when `command` uses `{service}`	The only services the action may target
`command`	array of string	Yes	An argument list, run without a shell. The only placeholder is `{service}`, filled with the service of the work item, which MUST be one of `services`; otherwise the action is not a candidate

The rules of the core, section 9.2, apply as written: no shell, no parameters from outside the allowlist, a separate credential for acting. `command` is required by this profile; the shared `declared-actions` schema cannot require it, because it does not know which profile a file is for.

6. The HTTP binding

The core's seven operations (core, section 8) travel as seven HTTPS calls, all made by the capability. From 0.2 this binding is shared with the Hosted profile, which uses it unchanged; the profile is named at enrolment. The diagram shows one whole hire; the examples page walks through the same hire with every request and response shown.

One hire over the HTTP binding



The seven calls of the Resident binding in the order they happen, with the schema name on each arrow and the receipts the capability writes on the left. Steps are numbered 0 to 8, as in the walkthrough.

All paths are relative to the base address the hirer gives the capability, written `<base>` below, for example `https://<tenant>.hub.example.com/saacip/`. Requests and responses are JSON. After enrolment, every request carries `Authorization: Agent <credential>`. The scheme name is `Agent`, so it never collides with `Bearer` tokens used by people.

Operation	Method and path	Request schema	Success	Response schema
Enrol	`POST <base>/enrol/`	`enrol-request`	201	`enrol-response`

Operation	Method and path	Request schema	Success	Response schema
Heartbeat	`POST <base>/heartbeat/`	`heartbeat`	200	`heartbeat-response`
Report	`PUT <base>/work-items/<ref>/`	`report`	201 created, 200 updated or unchanged	`report-response`
Request approval	`POST <base>/approvals/`	`approval-request`	201 new, 200 repeat	`approval`
Collect inbox	`GET <base>/inbox/`		200	`inbox`
Acknowledge	`POST <base>/inbox/ack/`	`ack`	200	`ack-response`
Send statement	`POST <base>/statement/`	`statement-request`	200	`statement-response`

6.1 The messages

Each message is defined by its schema, and every member has a description there. The members an implementer most needs to know:

Message	Members
`enrol-request`	`enrolment_token`; `protocol_versions`, preferred first; `profile`; `version` of the capability; `terms` (core, section 7.2), which carry the public key, the environment and the connector manifests
`enrol-response`	`protocol` and `profile` chosen; `agent_id`, the hub's identifier for this capability in this hire; `agent_token`, the credential, shown once; `settings` (core, section 6.3); `hub_public_key` *(0.2)*, the key the hub's attestations verify with
`heartbeat`	`status`, `ok` or `degraded`; `connectors`, each with `id`, `kind`, `ok`, `detail`; `counters`, the current track record; `receipt_head`, the hash of the latest receipt line; `hire`, the capability's view of its hire
`heartbeat-response`	`settings`; `release_requested`
`report`	For a new ref: `subject`, `description_html`, optional `priority_level` (1 is most urgent). For an open ref: `note`, which is added to the item. For a closed ref: `note`, which reopens the item and clears its outcome. A note alone for an unknown ref is answered with `not-found`. *(0.2)* `handoff`, an object `{"role": ..., "reason": ...}`, hands the item to a role (core, section 6.5); `origin`, an object `{"agent_id": ..., "ref": ..., "hub_ref": ...}`, says a new item was taken up from a hand-off
`report-response`	`hub_ref`, the hub's own reference, for example a ticket number; `status` in the hub's words; `created`; `updated`; `reopened`; `handed_off` *(0.2)*, true when the hub accepted a hand-off
`approval-request`	`action_id`, chosen by the capability; `ref` of the work item; `action`, the id of the declared action; `summary`
`approval`	The same, plus the hub's `id`, `status` (`pending`, `approved`, `rejected`), `decided_by`, `decided_at`
`inbox`	`approvals`, those decided and not yet acknowledged; `outcomes` (core, section 7.4); `work_items` *(0.2)*, items handed to a role this hire holds, each with `hub_ref`, `subject`, `description_html`, `priority_level`, `origin` and `role`; `release_requested`
`ack`	`approvals`, the ids applied; `outcomes`, the refs applied; `work_items` *(0.2)*, the `hub_ref`s of handed-off items taken up

Message	Members
`ack-response`	How many of each were marked delivered
`statement-request`	`statement` (core, section 7.5); `final`, default false
`statement-response`	`status` of the hire; `attestation` *(0.2)*, the hub's signed record, always with an accepted final statement (core, section 7.7)

Hub-side operations for people (creating hires, issuing a new enrolment token, releasing, revoking, recording verdicts, deciding approvals, defining roles and assigning them to hires) are part of the hub, not of the binding. Each hub documents its own; the reference hub documents its people API in its README.

6.2 Example: enrolment

```
POST /saacip/enrol/ HTTP/1.1
Host: acme.hub.example.com
Content-Type: application/json

{"enrolment_token": "enrol_...", "protocol_versions": ["saacip/0.2", "saacip/0.1"],
 "profile": "resident", "version": "1.0.0", "terms": { ... }}
```

```
HTTP/1.1 201 Created
Content-Type: application/json

{"protocol": "saacip/0.2", "profile": "resident",
 "agent_id": "5b1b273c-847f-4ecc-8dc7-0c2f5246cecd", "agent_token": "agent_...",
 "settings": {"autonomy_level": 2, "hire_kind": "trial", "trial_days": 14, "poll_seconds":
60},
 "hub_public_key": "..."}
```

6.3 Example: an error

Every error is problem details (core, section 8.7).

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{"type": "https://thesoftwarelens.com/saacip/0.1/errors/#unsupported-version",
 "title": "Unsupported version", "status": 400,
 "detail": "None of the offered protocol versions is supported.",
 "supported": ["saacip/0.2", "saacip/0.1"]}
```

7. Operating rules

1. There MUST be only one running instance per hire, on one state store; a deployment SHOULD stop the old instance before starting a new one.
2. After a hire ends, the capability MUST stay idle (or exit) and MUST NOT enrol again unless given a new enrolment token.
3. Heartbeats SHOULD be sent at least every 5 minutes; the inbox SHOULD be collected every `poll_seconds`.

4. The hub's work item for a report is often a help desk ticket; `hub_ref` is then the ticket number, and a later report with the same ref becomes an internal note on it.
5. The retry, unreachable-hub and refused-credential rules of the core (sections 8.3, 8.4 and 8.6) apply to every call in this binding.
6. *(0.2)* A handed-off item in the inbox is untrusted input: the capability reads it as data, takes it up under its own ref with `origin`, and applies its own terms and declared actions to it as to anything it found itself.
7. *(0.2)* On receiving an attestation, the capability SHOULD verify it with the `hub_public_key` from enrolment and write `attestation_received`; a failed verification is written in that receipt's `body` and reported to the hirer by the hub's own means, since the hire has ended.

8. Conformance

A capability conforms to this profile when it conforms to the core, keeps the four rules of section 3, carries connector manifests in its terms, runs declared actions as section 5 says, and makes the seven calls of section 6 with messages that validate against their schemas. A hub conforms when it answers those seven calls as section 6 and the core say, with responses and errors that validate against their schemas. A conformance claim names the version and the profile: "SaaCIP 0.2, Resident profile, capability". A 0.1 Resident capability conforms to 0.2 as it stands; the 0.2 additions are optional members it may ignore, except that it will not verify attestations or take up hand-offs.

Appendix A. Change log

Version	Date	Changes
0.1 draft	30 October 2026	First public draft, split out of the core document: residence rules, connectors and records, Resident declared actions, HTTP binding with a schema per message, operating rules
0.2 draft	6 November 2026	Outcome values of this profile named and mapped onto the core values (3.1); credentials (3.2); the binding shared with the Hosted profile; `hub_public_key` in the enrol response; `handoff` and `origin` in a report, `handed_off` in its response; `work_items` in the inbox and the ack; `attestation` in the statement response; operating rules 6 and 7

Comments are welcome. The views here are the author's own.

Important notice. This specification is part of my ongoing research into the Agentic Economy and reflects my thinking at the time of writing. It is shared in good faith, for information and discussion. It is not legal, regulatory, technical, financial or other professional advice, and it is not a verdict on how the EU AI Act or any other law applies to any organisation, system or situation. Laws, guidance, technology and my own research all move on, and others may reach different conclusions. The examples, designs, diagrams and technology choices show one possible approach. Before relying on, adapting or applying any part of it in your own situation, you must test and verify it, design it for your own architecture, and take independent expert advice, including from qualified lawyers and from your own IT, security and data protection specialists. Any decision to act on it, and its consequences, rest with you. To the fullest extent permitted by law, the author accepts no responsibility or liability for any loss, damage or consequence arising from any use of, or reliance on, this specification, which is provided without any warranty of accuracy, completeness or fitness for a particular purpose. Reading it does not create a professional or advisory relationship with the author. The views expressed are the author's own.