Agent Correction Format (ACF) -- v0.1

An open proposal by ProveCart. Status: proposal, version 0.1, published 2026-11-04. Canonical page: https://provecart.com/standards/correction-format Reference implementation: the ProveCart verification API (https://api.provecart.com), where this format is live today.

Name note: "ACF" in this document means only the Agent Correction

Format defined here. It is not affiliated with ACF Standards

(the Agent Certification Framework) or any other body using the

same initials.

1. Purpose

Payment and booking standards for AI agents can prove *who* an agent is and *that a purchase was approved*. They do not define what a checker should say to an agent when the thing the agent built does not match what the customer asked for. In practice, a stopped agent gets a bare refusal and cannot repair its own work.

ACF defines that missing message: a machine-readable correction payload that travels with a negative or uncertain verdict and tells the agent, per failed dimension, what is wrong, what was expected, what was submitted, and what action would fix it -- plus the rules for retrying, for escalating to the buyer, and for proving later what happened.

The format is deliberately small. It describes the mismatch only. It never describes the checker's internal method: no scores, no thresholds, no matching internals. Expected and actual values only.

ACF is published as an open proposal. Any checker, agent framework, or store may emit or consume it. ProveCart invites adoption, implementation reports, and corrections to this document; the format is versioned (Section 12) and changes will be published here first.

2. Roles and terms

- Instruction reference -- the customer's instructions, captured

as a signed, tamper-evident record (in the reference implementation, a *mandate*). It carries, as applicable: item or service description, brand, category, variant constraints, quantity limit, price ceiling and its basis, currency, allowed and excluded merchants, traveler or guest names, dates, times, party size, fare and cancellation terms, and an expiry.

- Subject -- what the agent actually built: a cart, a booking,

or a reservation, submitted for checking before payment capture (and, for travel, before ticket issuance).

- Checker -- the service comparing subject against instruction

reference and returning a verdict.

- Verdict -- one of MATCH, REVIEW, BLOCK (Section 3).

- Dimension -- one named aspect of the comparison (for example

price, item_match, traveler_names).

- Correction -- one entry of the corrections array (Section 4).

- Evidence record -- the sealed record of one check (Section 8).

3. The verdict object

A check returns a single JSON object:

verdict -- always / MATCH, REVIEW, or BLOCK

results -- always / Per-dimension results (below)

record_id -- always / Identifier of the sealed evidence record for this check

verdict_token -- always / Signed token binding this verdict to its record (Section 8)

corrections -- only when non-empty / The ACF correction array (Section 4)

max_attempts -- when corrections is present / Suggested retry budget for the correction loop; the reference value is 3

subject hash (e.g. cart_hash) -- implementation-defined / Hash of the subject as checked

Verdict semantics:

- MATCH -- the subject conforms to the instruction reference.

No corrections are emitted.

- REVIEW -- the checker cannot confirm conformity, or a fix

would need the buyer's decision (Section 6). A human or the buyer decides; the agent must not silently proceed.

- BLOCK -- the subject positively violates the instruction

reference. Payment (or issuance) must not proceed on this subject.

Each entry of results is:

```
{
  "dimension": "price",
  "status": "hard_fail",
  "reasons": ["total on the mandate basis (grand_total) exceeds the ceiling: compared figure 378
.0 ... vs ceiling 300.0 USD."],
  "details": { }
}
```

- status is one of pass, soft_fail, hard_fail.

hard_fail dimensions drive BLOCK; soft_fail dimensions drive REVIEW.

- reasons are human-readable strings generated from the same

structured values the correction carries.

- A failing dimension the agent cannot fix by editing the subject

carries "correctable": false (Section 5).

4. The corrections array

Present only on REVIEW or BLOCK, and only when at least one failing dimension is agent-correctable. Each entry:

dimension -- string / The failed dimension this correction belongs to

action -- string / The fix to perform, from the closed vocabulary of Section 4.1

reason -- string / The human-readable reason, identical to the dimension's reason text

expected -- any / The value the instruction reference requires (scalar, or an object such as {"size": "65 inch", "unit": "inch"})

actual -- any / The value found in the subject

item -- string, optional / The subject line item concerned, named exactly as submitted

field -- string, optional / The subject field concerned (booking and reservation dimensions)

Severity is expressed structurally in v0.1, not by a dedicated field: a correction inherits the severity of its dimension's status (hard_fail = blocking, soft_fail = review) under the response's overall verdict. A per-correction severity field is reserved for a future version; consumers must not require it.

4.1 Action vocabulary (closed list, v0.1)

Order (cart) subjects:

remove_item -- item_match / Remove a line item the instruction never asked for

replace_item -- item_match, category, brand, specification / Replace the named item with one that conforms

change_variant -- variant / Change the named attribute (color, size, capacity, model) to the expected value

reduce_quantity -- quantity / Reduce total quantity to at most the expected value

reduce_total -- price / Bring the compared total to at most the ceiling (expected)

change_currency -- currency / Re-price the subject in the expected currency

Booking subjects:

correct_traveler_name -- Correct the traveler name to the expected name

change_airport -- Use the expected airport (no silent substitution, including within a metro)

change_travel_date -- Move the leg or stay to the expected date

change_departure_time -- Move the departure to the expected time

add_itinerary -- Add the missing leg or stay

rebuild_itinerary -- Rebuild the itinerary so connections and sequence conform

change_cabin -- Book the expected cabin / fare class

change_fare_terms -- Book terms matching the expected refundability / change rules

add_hotel_stay -- Add the missing hotel stay

change_room -- Change the room selection (type, occupancy, dates) to the expected value

reduce_total -- Bring the all-in total to at most the ceiling

change_currency -- Re-price in the expected currency

Reservation subjects:

change_provider -- Use the expected provider

change_location -- Use the expected location / branch

change_date -- Move the reservation to the expected date

change_time -- Move the start to the expected time (a tolerance applies only where the instruction reference states one)

change_party_size -- Set the party size to the expected value

correct_reservation_name -- Correct the name on the reservation

reduce_deposit -- Bring the deposit to at most the expected amount

reduce_total -- Bring the amount to at most the ceiling

change_cancellation_terms -- Book cancellation terms matching the expected rules

New actions may be added in minor versions; consumers must ignore actions they do not recognize and treat the correction as not auto-applicable.

5. Non-correctable failures

Some failures cannot be repaired by editing the subject: an instruction reference that is missing, unsigned, tampered with, or expired; an excluded merchant; an exhausted standing budget. For these, the checker:

- emits no correction for the dimension, and
- marks the dimension's result "correctable": false,

so an agent does not loop retrying an unfixable subject. In the reference implementation the non-correctable dimensions are mandate, mandate_signature, mandate_expiry, merchant, standing_budget, and engine_error (a checker-side failure, which fails safe to REVIEW).

6. The retry contract

1. A corrected subject is submitted as a **new check against the

same instruction reference**. The resubmission carries previous_record_id, the record_id of the attempt it supersedes. Every attempt is sealed as its own evidence record; earlier verdicts are never rewritten. The chain of attempts -- refusal, correction, re-check -- is itself the dispute evidence.

2. max_attempts in the verdict object is the checker's suggested

budget for the loop (reference value 3). Exhausting it is not a verdict; the agent stops and escalates.

3. Buyer-approval escalation. An agent may auto-apply only

corrections that bring the subject *into* conformity with the signed instruction reference. Any change that would go beyond what the buyer authorized -- paying more than the ceiling, accepting a materially different product, name, date, or terms than instructed -- is never an auto-fix: the check returns REVIEW, and the agent must ask the buyer and proceed only with a fresh instruction or explicit approval.

4. A corrected subject that verifies MATCH completes the loop.

The pair (or chain) of records is the "sale saved" evidence: proof the mistake was caught before payment, and fixed.

7. Provenance of the instruction reference

Corrections are only meaningful against an instruction the buyer actually gave. In the reference implementation:

- the instruction reference is created as a signed record

(HMAC-SHA256 over its contents, in the reference implementation), identified by a mandate_id;

- the reference travels with the subject at check time;
- the checker verifies the signature before checking

(a failed verification is the non-correctable dimension mandate_signature);

- the correction payload exposes only the values the mandate

holder already gave the agent: expected values are the buyer's own stated requirements, never checker internals.

Implementations may substitute an equivalent signed-instruction scheme (for example a Verifiable Intent / AP2 mandate) provided the signature is verified at check time and the reference is bound to the evidence record.

8. The sealed evidence pointer

Every check -- pass, review, or block -- is written to a tamper-evident, hash-chained evidence log before its verdict is relied on. The verdict object points back to it:

- record_id identifies the sealed record, which binds the

instruction reference, the subject as submitted, the results, the corrections, and the verdict.

- verdict_token is a short-lived signed token over the verdict

payload. A verdict token is single-use and hold-bound: it authorizes at most one downstream action (one payment capture, one ticket issuance) against exactly one record, and at action time the token's instruction reference and subject hash must equal the values stored on the hold being acted on -- checked against the hold record itself, never against values re-supplied in the action request. A corrected subject produces a new record and a new token; tokens never carry over.

- Records export as dispute bundles: what the customer asked for,

what the agent built, what was checked, and what was decided, in order, verifiable against the chain.

Gateways and booking systems integrating a checker must enforce the single-use, hold-bound rule on their side. It is part of the integration contract, not an implementation detail.

9. Worked example (from the reference implementation)

A buyer instructed: Sony WH-1000XM5 headphones, at most 1, ceiling $300. The agent built a cart with two BassPro Q9 headphones at $378 total. The checker returned BLOCK with three corrections (captured verbatim from the reference implementation's response; the per-dimension results array is elided here):


```
{
  "verdict": "BLOCK",
  "record_id": "...",
  "verdict_token": "...",
  "max_attempts": 3,
  "corrections": [
    {
      "dimension": "price",
      "action": "reduce_total",
      "reason": "total on the mandate basis (grand_total) exceeds the ceiling: compared figure 3
78.0 (merchandise 378.0 + tax 0.0 + shipping 0.0 + fees 0.0 = 378.0) vs ceiling 300.0 USD.",
      "expected": 300.0,
      "actual": 378.0
    },
    {
      "dimension": "quantity",
      "action": "reduce_quantity",
      "reason": "total quantity 2 exceeds mandate max_quantity 1",
      "expected": 1,
      "actual": 2
    },
    {
      "dimension": "item_match",
      "action": "replace_item",
      "item": "BassPro Q9 Wireless Headphones",
      "reason": "item conformity is indeterminate or partial for: BassPro Q9 Wireless Headphones
 (coverage in the indeterminate band routes to REVIEW)",
      "expected": "Sony WH-1000XM5 wireless noise cancelling headphones",
      "actual": "BassPro Q9 Wireless Headphones"
    }
  ]
}
```


The agent's conforming repair: replace the item with the instructed Sony headphones, quantity 1, total within $300 -- resubmitted as a new check with previous_record_id set to the blocked attempt's

record_id.

10. Conformance

- ACF emitter (checker) -- returns the verdict object of

Section 3; emits corrections per Sections 4-5 on every correctable failure; marks non-correctable dimensions per Section 5; seals every check and honors Section 8's token rule.

- ACF consumer (agent) -- reads corrections before escalating;

auto-applies only in-instruction fixes; escalates any fix that would exceed the buyer's authorization to the buyer (Section 6); honors max_attempts; never retries a dimension marked "correctable": false.

- ACF integrator (gateway / booking system) -- enforces the

single-use, hold-bound verdict-token rule on its side (Section 8) and refuses a second capture or issuance against one record.

11. Companion proposals (reserved, not specified)

Four adjacent gaps surfaced while defining this format. They are named here so the territory is marked; none is specified in v0.1, and this section creates no conformance requirements.

- Behavior-credential field for agent cards. Agent cards

declare identity and capabilities; there is no standard field by which an agent declares a *behavior* attestation -- who examined its buying accuracy, on what exam version, with what score, status, and expiry. A companion proposal would define that field for card formats such as A2A's Agent Card.

- Human-approval proof packet. When an agent pauses and

asks its buyer to approve a deviation, there is no standard packet for what the buyer is shown: the instruction, the deviation, the correction offered, and the sealed record behind all three. A companion proposal would define the packet, so approval moments are legible and auditable.

- Score version fingerprint. An exam score is meaningless

without the version of the exam that produced it. A companion proposal would define a short fingerprint (exam id, version, task-bank hash) that travels with any published score, so scores from different exam versions are never compared as if they were the same test.

- The not-call vocabulary. Guidance today tells agents what

to do; there is no shared vocabulary for when an agent should *not* call a tool at all -- when the conforming action is to stop, ask, or abstain. A companion proposal would name those states so checkers, exams, and agent frameworks can refer to the same decision.

12. Versioning

This document is version 0.1. The canonical page (https://provecart.com/standards/correction-format) always serves the current version; every version is also published as an immutable file beside it, and an immutable file never changes after publication.

- Minor versions (v0.2, ...) may add actions to the Section 4.1

vocabulary, add optional fields, and clarify wording. They may not remove or rename actions or fields, change verdict semantics, or change the retry and escalation rules of Section 6.

- Major versions are required for anything a minor version may

not do, and ship as a new immutable file with a migration note in the changelog.

- Changes are proposed in the open: an issue or pull request

against the spec in the ProveCart repository, or a comment in the standards discussion where this format was filed. Proposals are accepted, declined, or deferred with reasons recorded in the changelog. A proposal backed by a second independent implementation carries the most weight.

- Errata -- fixes that change no normative content -- are applied

to the current version, dated in the changelog, and do not change the version number.

---

13. Licensing

The point of publishing a format is that other people build to it, so the terms are simple.

- The document. This specification text may be copied and

shared freely, with attribution to ProveCart.

- The format is free to implement. ProveCart commits that

any patent claim it owns or controls that is *necessarily infringed* by implementing this format as published -- an essential claim -- is licensed royalty-free to anyone, for the purpose of implementing the format. No implementer needs ProveCart's permission, account, or service to conform.

- Scope. The commitment covers the format: the verdict

object, the corrections array, the code and action vocabularies, and the fields defined in this document. It does not license ProveCart's verification service, checker, certification exam, evidence store, or brand, and it does not extend to patent claims that are not essential to implementing the format.

Version stamp Agent Correction Format (ACF) · Version 0.1 · Status: open proposal -- current · Published 2026-11-04 · Editor: ProveCart · Canonical page: https://provecart.com/standards/correction-format · Reference implementation: the ProveCart verification API (https://api.provecart.com), live · Implementations known: 1 (the reference implementation) · Worked examples, implementer guide, and test vectors are published with this version.

Changelog

- v0.1 -- 2026-11-04. First publication: verdict object,

corrections array with the closed action vocabulary for cart, booking, and reservation subjects, non-correctable dimensions, the retry contract with buyer-approval escalation, provenance of the signed instruction reference, and the sealed evidence pointer with its single-use, hold-bound token rule.

- Errata -- 2026-11-05. Restored the end of Section 10 and

Sections 11-12, which were cut off in the first publication; added this version stamp and changelog. No normative change.

- Licensing -- 2026-11-04. Added Section 13: the format is

royalty-free to implement (an essential-claims commitment in RAND-Z style), per the founder's decision of 2026-11-04. No change to the format itself.