Conformity Corrections for Agent Checkout -- ACF v0.2

Agent Correction Format (ACF), version 0.2.0 -- DISCUSSION DRAFT.

Status: discussion draft, published for implementer feedback. This is not a request for adoption into any dev.ucp.* namespace, and no adoption or endorsement is implied. The extension described here is self-hosted under its publisher's namespace, as UCP's contributing guide prescribes for vendor extensions.

Publisher of this draft: ProveCart (author and reference implementation). The format is vendor-agnostic in substance: any checker can conform, and nothing in conformance requires the publisher's API, scoreboard, or any credential issuer (Section 14).

License. This document and the JSON Schema 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 the format as published -- an essential claim -- is licensed royalty-free to anyone, for the purpose of implementing the format (a RAND-Z style commitment). The commitment is stated in full in Section 16 and identically in Section 13 of ACF v0.1. The commitment covers the format only; 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. (Decided by the publisher 2026-11-04.)

What changed from v0.1. Version 0.1 defined a standalone verdict object with a parallel corrections array. Independent review against the UCP specification showed that UCP checkout already carries the channel this format needs, so v0.2 is reshaped onto UCP's own primitives and its delta is deliberately smaller:

- Failing dimensions are reported as entries in the checkout's

existing messages[] array, using a namespaced code vocabulary (Section 5) instead of a parallel array. The host's JSONPath path replaces v0.1's dimension/field strings.

- expected and actual values, the closed action vocabulary, the

retry budget and the sealed-evidence pointer -- the parts messages[] does not have -- ride as extension fields (Section 5) and one namespaced conformity object on the checkout (Section 4).

- The extension is renamed from its v0.1 working name to

com.provecart.conformity ("verification" collides with payment authentication and identity concepts in the host ecosystem).

- v0.1 remains the version the reference implementation emits today

at its own API. v0.2 is the discussion draft proposed for the host protocol. Where the two differ, the delta is stated in this document, not hidden.

Normative schema: acf-v0.2.schema.json (JSON Schema Draft 2020-12), published alongside this document. Conformance fixtures: fixtures/ in this package. Profile declaration example: profile-declaration-example.json.

The key words MUST, MUST NOT, SHOULD and MAY are to be read as in RFC 2119.

---

1. Purpose, stated against what already exists

This format does not claim an empty field, and it is written against what the host standards already provide:

- UCP checkout already reports failures. Every checkout response

carries a messages[] array -- each entry has a type, a code, a severity, an optional JSONPath path to the exact field, line item or occurrence, and human-readable content -- and a status lifecycle (incomplete, requires_escalation, ready_for_complete) that already encodes "fix this and resubmit," "hand this to the buyer," and "proceed."

- AP2 already binds signed mandates. AP2 provides signed

authorization artifacts; in UCP the buyer's instruction is the AP2 Intent Mandate, carried by dev.ucp.common.payment.ap2_mandate.

What those mechanisms do not carry, and what this extension adds, is narrow: when an independent check compares a built checkout against the buyer's signed instructions and finds a mismatch, the repair guidance is today free text a person can read and a program cannot reliably act on. This extension adds, on top of messages[]:

1. the expected and actual values, so a repair is a

computation, not an interpretation;

2. a closed action vocabulary, so the class of fix is enumerable

and testable;

3. a retry budget, so the correction loop terminates by rule;

4. one conformity object on the checkout carrying the verdict and

a pointer to a sealed evidence record, so the outcome is auditable after the fact.

Framing, stated plainly: this document is a profile for how a checker reports conformity against an AP2 Intent Mandate. It defines the reporting shape. It does not define the mandate, the comparison, or the checkout.

2. Roles, the emitter, and where the instruction comes from

- Business. In UCP, checkout responses come from the Business.

The conformity object and conformity-coded messages are therefore emitted by the Business, which runs or relays the check -- either operating a conforming checker itself or relaying the output of one it has engaged (for example, its checkout provider). An extension field with no sender in the protocol is not a proposal; this section names the sender.

- Checker. The component that compares the checkout, as built,

against the buyer's signed instructions and produces the verdict, the messages and the sealed evidence record. A checker may be operated by the Business, by a checkout provider, or by an independent verification service engaged for the transaction. The format constrains the checker's *output*, not its ownership.

- Platform / agent. The buyer's agent and its runtime: the

consumer of the messages and the conformity object. It applies in-authorization repairs, escalates to the buyer when the host status or a message's severity requires it, and stops at the retry budget.

- Buyer. The human or organization whose signed instructions

authorize the purchase. The only party who can widen an authorization.

- The instruction. In UCP terms, the AP2 Intent Mandate. The

conformity object binds to it by digest (Section 4); the format never carries the mandate itself. *Ingestion note, stated honestly:* the reference implementation today verifies its own HMAC-signed instruction

references and treats AP2 / Verifiable Intent mandates as substitutable sources; consuming the AP2 Intent Mandate artifact directly is a roadmap item, not built, and is not claimed anywhere in this document.

3. Extension declaration, discovery and negotiation

- Extension id: com.provecart.conformity, versioned with

semantic versioning. This document is version 0.2.0: pre-1.0, signaling a discussion draft whose shape may still change.

- Extends: dev.ucp.shopping.checkout and

dev.ucp.lodging.booking.

- Declaration. Each side declares the extension in its profile

with {version, extends, spec, schema} -- the spec URL and the JSON Schema URL a counterparty needs to implement against it. profile-declaration-example.json in this package shows both the business and the platform side.

- Negotiation. The extension is in force for a checkout only when

both sides' profiles declare a common version. Where the versions differ, the lower common version's vocabulary governs; a party that does not recognize a code or action behaves as in Section 5.3.

- Absence is safe. A checkout with no conformity object is an

ordinary checkout: no verdict is implied, and in particular the absence of a check MUST NOT be presented or treated as a pass. A party that did not negotiate the extension ignores the namespaced object and any conformity-coded messages without error.

4. The conformity object

One namespaced object on the checkout, under the single key com.provecart.conformity -- nested under one key the way the AP2 extension nests under ap2, so canonicalization and signature-exclusion rules have exactly one place to apply.

verdict -- MATCH, REVIEW or BLOCK (Section 6).

record_id -- Identifier of the sealed evidence record for this check. Also the idempotency anchor: a replayed check of the same checkout under the same instruction reference returns this same record_id; a verdict is never silently re-issued as a new record.

verdict_token -- Short-lived signed token binding the verdict to its record. Single-use and hold-bound: it authorizes at most one downstream action (one payment capture, one ticket issuance) against exactly this record, verified against the hold record itself, never against values supplied in the request. Required when the object is presented to authorize a downstream action.

evidence_url -- Where the sealed record can be retrieved by an authorized party. Retrieval is access-controlled; public exports are redacted (Section 10).

mandate_ref -- Reference to the buyer's signed instructions: an identifier and scheme, and a digest of the exact instruction artifact checked. Never the instructions themselves.

subject_digest -- Canonical digest of the checkout as checked. With mandate_ref.digest it binds the verdict to exactly one checkout state and one instruction version; any change to either requires a new check.

issued_at / expires_at -- The verdict's validity window. After expires_at the verdict MUST NOT be acted on; a stale verdict presented downstream MUST be refused and the checkout re-checked.

previous_record_id -- The record this submission supersedes, on a corrected resubmission (Section 7).

max_attempts -- The retry budget for the correction loop (reference value 3).

Replay rule: a verdict is a statement about one digest pair. A checkout that differs from subject_digest, an instruction that differs from mandate_ref.digest, or a time after expires_at makes the verdict void, not merely stale -- the remedy is a new check, and the new check seals a new record linked by previous_record_id where it is a resubmission.

5. Messages: the code vocabulary and the extension fields

5.1 Shape

Each failing dimension produces one entry in the checkout's messages[], in the host's own shape -- type, code, severity, path (JSONPath into the checkout object), content -- with the extension's additions carried as extra fields on that same entry:

code -- extension / From the vocabulary in 5.2: com.provecart.conformity.<failure_code>, one code per failing dimension.

expected -- extension / The value the instruction requires. Subject to the disclosure rules of Section 10.

actual -- extension / The value found in the checkout as built. Same disclosure rules.

action -- extension / The fix to perform, from the closed vocabulary in 5.4.

uncertain -- extension / true when the checker could not positively determine conformity (an indeterminate determination); false when the violation is positively established.

requires_buyer_approval -- extension / true when no repair within the buyer's existing authorization is available. Section 8 governs. The host severity for such a message is requires_buyer_review in any case; this flag states the same rule per correction.

content remains the human-readable rendering of the same structured values; where the two disagree, the structured fields govern and the disagreement is a defect in the emitter.

5.2 Code vocabulary (initial set)

One code per failing dimension, drawn from the reference implementation's result codes, lowercased under the namespace:

price_over_ceiling, price_below_floor, quantity_over_max, quantity_below_min, item_mismatch, item_indeterminate, item_excluded, variant_mismatch, currency_mismatch, merchant_excluded, date_mismatch, date_outside_window, time_outside_tolerance, party_size_mismatch, name_mismatch, cabin_mismatch, fare_terms_mismatch, deposit_over_limit, cancellation_terms_mismatch, mandate_expired, mandate_signature_invalid, standing_budget_exceeded, engine_error.

The vocabulary is versioned with the extension. New codes are added by minor version; a code's meaning never changes within the vocabulary.

5.3 Unknown codes and actions

A consumer that meets a code it does not recognize, or an action outside the vocabulary it implements, MUST NOT auto-apply the entry, MUST NOT guess its meaning, and MUST NOT treat it as a pass. The conformity object governs the outcome; the unknown entry escalates to the buyer. Fixture f08 in this package pins this behavior.

5.4 Action vocabulary (closed)

Carts: remove_item, replace_item, change_variant, reduce_quantity, reduce_total, change_currency. Bookings: correct_traveler_name, change_airport, change_travel_date, change_departure_time, add_itinerary, rebuild_itinerary, change_cabin, change_fare_terms, add_hotel_stay, change_room,

reduce_total, change_currency. Reservations: change_provider, change_location, change_date, change_time, change_party_size, correct_reservation_name, reduce_deposit, reduce_total, change_cancellation_terms.

5.5 Non-correctable failures

Some failures have no repair by editing the checkout, and a conforming checker MUST NOT emit an action for them: an expired or invalidly-signed mandate, an excluded merchant, a standing budget or standing instruction that itself forbids the purchase, and a checker engine error. Their messages carry severity unrecoverable (or requires_buyer_review where a fresh buyer decision is the remedy), and the outcome escalates. A checker engine error fails safe to REVIEW and is itself sealed in the evidence record; an error is never a pass.

6. Verdict semantics and the mapping to the host lifecycle

The verdict is the checker's summary; the host status is what the platform acts on. They correspond as follows (this table is the normative mapping):

Checker verdict -- Host checkout state / Platform action

MATCH -- status ready_for_complete, no conformity-coded messages / Complete the checkout

BLOCK, with a fix -- status incomplete + message(s), severity recoverable, with path / Modify inputs per the extension fields, resubmit as a new check

REVIEW -- status requires_escalation + message(s), severity requires_buyer_review / Hand off to the buyer (via continue_url); capture MUST NOT proceed

Non-correctable failure -- message(s), severity unrecoverable / New inputs, a new instruction, or hand off; no edit of this checkout repairs it

Rules that bind the mapping:

- Payment capture (and, for bookings, ticket issuance) may proceed

only on MATCH / ready_for_complete.

- REVIEW is a stop, not a soft pass: where any dimension is

indeterminate, or any repair needs the buyer, the outcome is REVIEW and the decision moves to the buyer.

- BLOCK means the checkout as built cannot complete; the messages

say what would have to change, and a changed checkout is a new checkout state and a new check.

7. The retry contract

Status note (updated 2026-11-04): the security verification

this section awaited is closed. An independent code audit

had flagged that previous_record_id must be validated for

ownership and mandate binding; the reference implementation now

validates the link at intake (the prior record must exist,

belong to the same API key and the same mandate, and still head

its attempt chain), enforces the retry budget of 3 attempts per

chain rather than merely advertising it, and refuses capture

against superseded verdicts. The attack battery (foreign record,

cross-mandate record, fabricated record id, over-budget chain,

superseded-token capture) passes against the production code,

which was deployed and hash-verified on 2026-11-04.

1. A resubmission after repair is a new check: a new checkout

state, a new subject_digest, a new sealed record, carrying previous_record_id so the attempt chain is auditable. Earlier records are never rewritten.

2. The chain keeps the same instruction reference. A resubmission

under different instructions is not a retry; it is a new purchase decision.

3. max_attempts bounds the loop (reference value 3). Exhausting it

is not a verdict: the agent stops, and the matter goes to the buyer or the merchant. A conforming consumer MUST NOT loop past the budget, and a conforming checker MUST NOT raise it silently mid-chain.

4. Only repairs within the buyer's existing authorization are

applied automatically (Section 8).

8. Buyer approval and reauthorization

A repair is auto-applicable only when it brings the checkout into conformity with the authorization the buyer already gave: a lower total toward the authorized ceiling, the authorized date, the authorized item. A fix that would change the price, the terms, the traveler's or recipient's identity, the destination, or the scope of what the buyer authorized is never a correction -- it is a new decision. For such a fix:

- the message carries requires_buyer_approval: true and severity

requires_buyer_review, and the verdict is REVIEW;

- an agent MUST NOT apply it automatically, under any retry budget;
- "approval" means a new or amended signed instruction reference

against which a fresh check runs. It never means clearing a flag, and a checker MUST NOT accept an instruction widened by the agent itself.

9. Determinism

The verdict on a given checkout and a given instruction reference is a function of the two artifacts, not of a model's discretion:

- No model, assistant layer or heuristic may silently alter signed

intent or authorization -- not the buyer's ceiling, dates, items, parties or terms -- whether to clear a check or to block one.

- Ambiguity escalates: where conformity cannot be positively

determined, the outcome is REVIEW and the buyer decides.

- Only clear violations BLOCK. An engine error is REVIEW, sealed.
- Where an implementation uses a model anywhere in its pipeline, the

model may propose and the deterministic check disposes: a model MUST NOT be able to overturn a BLOCK, and MUST NOT run at all on a MATCH. (These are the reference implementation's live hard limits, stated here as conformance requirements for any checker claiming this format.)

10. Privacy and disclosure

A checker necessarily sees the checkout and the instruction it is checked against. The format constrains what travels and what is published:

- The conformity object and messages travel on the authenticated

checkout channel between the negotiating parties. They are not a public artifact.

- expected/actual are carried only where disclosure is safe

for the parties to that transaction. Where a value would expose information a party has no right to see (for example, the buyer's full ceiling visible to a party who should see only that the total exceeds it), the emitter omits the value and marks the message disclosure: "withheld_privacy"; the verdict and the action are unaffected.

- Public evidence is redacted evidence. Any publicly retrievable

form of a record excludes traveler and recipient names, addresses, and the full text of the buyer's instructions; it carries digests, verdicts, codes and outcomes. Sealed full records are retrieved under controlled access by parties to the transaction and their dispute processes.

- The format itself requires no retention beyond the evidence

record; retention periods are a deployment property and are out of scope, except that a record relied on for a dispute MUST remain retrievable for the dispute.

11. Evidence

Every check produces a sealed evidence record before its verdict is relied on: the instruction reference and digest, the checkout digest, the per-dimension outcomes, the corrections offered, the verdict, the token issued, and the attempt-chain link. Records are hash-chained in the reference implementation so later tampering with an earlier record is detectable, and export as dispute bundles. What hash chaining does and does not establish is stated without embellishment in Section 13.

12. Conformance

Emitter (checker side). A conforming emitter produces the conformity object and messages per Sections 4-6, binds them per Section 4, applies the consent rule of Section 8 and the determinism rule of Section 9, seals every check per Section 11, and redacts per Section 10.

Consumer (platform/agent side). A conforming consumer acts on the host status per Section 6, applies only in-authorization repairs, escalates on requires_buyer_review and on unknown codes or actions (Section 5.3), honors the retry budget (Section 7), and refuses expired verdicts (Section 4).

Fixtures and harness. This package includes eight fixtures (fixtures/, with a README): seven generated from the reference implementation's production code paths -- a clean MATCH, BLOCKs with repairs across a cart, a booking and a reservation, a REVIEW on an indeterminate item, a corrected resubmission that then matches, and a BLOCK on a non-correctable dimension -- plus one hand-authored fixture pinning unknown-code consumer behavior. validate_fixtures.py checks the v0.1 invariants on each capture and validates the v0.2 projection of each against acf-v0.2.schema.json; third parties run it with the Python standard library plus jsonschema. The reference implementation's own public test batteries (cart, booking and reservation suites) are linked from the fixtures README. A reference consumer sample -- a small platform-side script that acts on the conformity object against the reference API -- ships in this package as reference-consumer.py.

Second implementations. Conformance is defined by this document, the schema and the fixtures. An implementer needs nothing from the publisher: no API account, no scoreboard entry, no credential.

13. Known limitations -- what this does not claim

- Feasibility is not adoption. One live implementation shows the

format can be built and run. It does not show that anyone else wants it, will implement it, or should. That is what the discussion is for.

- Hash chaining is not independent custody. A hash-chained log

makes tampering detectable to whoever holds the chain head; it does not by itself prove the records are held by an independent custodian, and the reference implementation's evidence store does not yet have the durability hardening (write-ahead fsync before acknowledgement, a database-backed store) that a custody claim would need. That work is named here as outstanding, not implied done.

- Patent-pending is not a novelty opinion. The publisher has

patent applications pending relating to its implementation. That is a statement about filings, not evidence that any claim is novel, valid or enforceable, and it is not a clearance opinion on this format.

- The mandate source is a profile, not an ingestion. Section 2's

note stands: direct consumption of the AP2 Intent Mandate artifact by the reference implementation is roadmap, not built.

- **The retry link is validated; its security verification is

closed.** A resubmission's previous_record_id must reference an existing verification record of the same mandate under the same API key, must still be the head of its attempt chain, and must be within the chain's retry budget; a missing or foreign reference is refused as a sealed BLOCK (PREVIOUS_RECORD_NOT_FOUND) with no link recorded. Standing-budget accounting is chain-aware -- captured spend is final, and a superseded verdict can no longer be captured or issued against -- so a retry chain cannot reset or evade the mandate's standing budget. Section 7 states the contract.

- **Fit to lodging beyond the reference implementation's reservation

and booking checks is untested** by any second party at the time of writing.

14. Governance and neutrality

- The terminology of this document is role-based (Business, checker,

platform, buyer) and vendor-agnostic. The publisher is named as author and reference implementation, and in the extension's namespace -- which is where a vendor extension in its discussion phase lives. Nothing else in the format is the publisher's.

- Any checker can conform (Section 12). Any business can run or

relay its own. If the vocabulary graduates toward a dev.ucp.* home, the namespace is the venue's to settle; the publisher's namespace is a starting point, not a claim on the outcome.

- Changes to this draft are versioned; substantive changes increment

the minor version pre-1.0 and are recorded in the document's revision note.

15. Versioning

This document is ACF 0.2.0. It supersedes v0.1 as the *discussion* draft; v0.1 remains the emitted format of the reference implementation until a v0.2 emitter ships, and both documents stay published so the delta is inspectable. Pre-1.0, minor versions may reshape fields; the code and action vocabularies only grow within a major line, and a code or action, once published, never changes meaning.

16. Licensing

This specification text and the JSON Schema 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 the format as published -- an essential claim -- is licensed royalty-free to anyone, for the purpose of implementing the format (a RAND-Z style commitment). The commitment was decided by the founder on 2026-11-04 and is stated identically in Section 13 of ACF v0.1. No implementer needs ProveCart's permission, account, or service to conform. The commitment covers the format only -- the conformity object, the message code vocabulary, the extension fields, and the retry and consent rules defined here; 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 Conformity Corrections for Agent Checkout (Agent Correction Format, ACF) · Version 0.2.0 · Status: discussion draft -- current · Published 2026-11-05 · Editor: ProveCart · Canonical page: https://provecart.com/standards/correction-format · JSON Schema: https://provecart.com/standards/acf-v0.2.schema.json · Superseded discussion draft: v0.1, archived at https://provecart.com/standards/correction-format-v0.1 · Reference implementation: the ProveCart verification API (https://api.provecart.com), live (emits v0.1 today) · Implementations known: 1 (the reference implementation).

Changelog

- v0.2 -- 2026-11-05. Discussion draft reshaped onto the

host protocol's own primitives after independent review against the UCP specification: failing dimensions are reported as entries in the checkout's existing messages[] array under the namespaced code vocabulary com.provecart.conformity -- the extension is renamed from its v0.1 working name ("verification" collides with payment authentication and identity concepts) -- with the expected/actual values, the closed action vocabulary, the retry budget and the sealed-evidence pointer carried as extension fields and one namespaced conformity object on the checkout, instead of v0.1's parallel corrections array. Extends both dev.ucp.shopping.checkout and dev.ucp.lodging.booking. Framed as a profile for how a checker reports conformity against an AP2 Intent Mandate. The royalty-free essential-claims (RAND-Z) licensing commitment is stated at the head of this document and in Section 16 (decided 2026-11-04). Section 7's retry contract: the previous_record_id security verification closed on 2026-11-04 (intake validation + enforced retry budget, deployed and hash-verified in production). v0.1 remains the version the reference implementation emits; v0.2 supersedes v0.1 as the discussion draft only.

- 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 (v0.1) -- 2026-11-05. Restored the end of

Section 10 and Sections 11-12 of v0.1, which were cut off in the first publication; added the version stamp and changelog. No normative change. This errata entry is carried forward here so the repair stays in the record; the full v0.1 text, with its own stamp and changelog, is archived at the URL in the version stamp above.

- Licensing (v0.1) -- 2026-11-04. Added Section 13 to

v0.1: 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.

- Pre-filing consistency sweep -- 2026-11-05. Section 13's

retry bullet updated from "under security verification" to the closed state Section 7 records (the retry link validated for tenant and mandate binding; the retry budget enforced, not advisory), and the license wording aligned so the same statement appears in this document, the JSON Schema and the filing text. No normative change.