Independent Submission
Internet-Draft
Intended status: Informational
Expires: 12 November 2026

D. Borthwick
InsumerAPI
M. Msebenzi
Headless Oracle

11 May 2026

Verifiable Intent — environment.* Constraint Family
draft-borthwick-msebenzi-environment-state-00

Abstract

The Verifiable Intent (VI) mandate format authorises autonomous agents to act on behalf of human principals through cryptographically signed constraint instances bound into Layer 2 mandates. VI's existing constraint families describe properties of the transaction itself — what is being purchased, by whom, for how much, against which credential. They do not describe properties of the environment in which the transaction is executed: whether the venue is open, whether the source of funds is funded, whether other relevant external conditions hold at the moment of execution.

This document specifies the environment.* constraint family for the Verifiable Intent mandate vocabulary. It defines the membership criterion under which a constraint type qualifies as a member of the family, the family-wide vocabulary every member uses, the composition discipline by which family members compose with each other and with constraints from other families, the register discipline under which family-wide and per-type prose is written, the family-wide security considerations, and the IANA registry mechanics under which new family members are registered. It does not define any individual constraint type. Two reference type specifications (environment.market_state and environment.wallet_state) are referenced informatively in Appendix A.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 November 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

Autonomous agents executing financial transactions on behalf of human principals require cryptographic awareness of external state at the moment of execution. This requirement exists independently of the constraints that authorize what the agent may do. An agent holding a valid mandate to purchase a specific asset for a specific amount must additionally know, at execution time, whether the venue at which it intends to execute is open and whether the source of funds it intends to draw from still satisfies the conditions under which the mandate was issued. Neither question is answered by the mandate itself. Both must be answered, with cryptographic evidence, before execution proceeds.

This document defines a constraint family — environment.* — for the Verifiable Intent (VI) mandate vocabulary that addresses this class of requirement. It does not specify any individual constraint type in detail. Instead, it defines the membership criterion under which a constraint type qualifies as a member of the environment.* family, the vocabulary by which family members compose with each other and with constraints from other families, and the discipline under which new family members may be added by future revisions or by independent implementations.

1.1. Motivation

The Verifiable Intent specification [VI-CONSTRAINTS] defines mandate constraints under a single transactional namespace, mandate.*, with two sub-namespaces: mandate.checkout.* declares what the agent may purchase (allowed merchants, line items), and mandate.payment.* declares how the agent may settle (allowed payees, amount ranges, budgets, recurrence, payment-mandate reference). These transactional constraint families are sufficient to authorize an agent’s intended action and to bind that authorization cryptographically to a credential the agent presents at the moment of execution.

They are not sufficient to bind the agent's action to the state of the world at the moment of execution. An agent can present a valid mandate to purchase shares of a security listed on a venue that is currently halted; the mandate is cryptographically valid and the action is unauthorized in the real world. An agent can present a valid mandate to settle a payment in a stablecoin from a wallet that no longer holds sufficient balance; the mandate is cryptographically valid and the settlement will fail or, worse, succeed against an unintended source. These failure modes are not exotic edge cases. They are the autonomous-execution analog of well-documented races in financial markets and on-chain settlement, including market-execution events such as the 2010 Flash Crash and circuit-breaker session-boundary races, and on-chain TOCTOU exploits in the flash-loan-enabled class.

Existing transactional constraint families cannot close this gap because they describe properties of the transaction — what is being done, by whom, for how much, against what credential. The gap concerns properties of the environment in which the transaction is executed — whether the venue is open, whether the source is funded, and, by extension, whether other relevant conditions of the world hold at execution time. These properties are not under the control of any party to the transaction and cannot be asserted by the agent. They must be observed and signed by an entity outside the transaction whose identity and signing key are bound into the mandate at issuance time.

A constraint family is therefore the right primitive: each member type targets a specific environmental property, signed by an external attester, and verified at execution time before the transactional constraints are evaluated. This document defines the family. Specific member types are defined in their own specifications; reference implementations are cited in Appendix A.

1.2. Scope

This document defines:

(1) the membership criterion under which a constraint type qualifies as a member of the environment.* family;

(2) the vocabulary used by family members, including required and optional fields whose semantics are family-wide;

(3) the composition discipline by which family members compose with each other and with constraints from other families;

(4) the register discipline (RFC 2119 keyword usage, normative versus informative material) under which family-wide and per-type prose is written.

This document does not define:

(1) any individual environment.* constraint type. Two reference type specifications (environment.market_state and environment.wallet_state) are documented in their own working-group submissions and are referenced informatively in Appendix A;

(2) any algorithm for canonicalising signed attestation payloads. Per-type canonicalisation is the responsibility of each type specification;

(3) any specific signing algorithm. Algorithm agility is family-wide; per-type MUST-implement declarations are made by each type specification;

(4) any specific attestation issuer, attestation endpoint, or operational detail of any reference implementation;

(5) any change to the transactional constraint families (mandate.*) defined in [VI-CONSTRAINTS]. Family composition is the only surface at which environment.* interacts with transactional constraints, and this composition is unidirectional: environment.* constraints gate execution; transactional constraints are evaluated after environment.* constraints have passed.

The deliberate narrowness of this scope reflects the family’s design discipline: the document is the contract under which independent type authors and implementers can extend the family without coordinating with each other or with the authors of this document, provided they satisfy the membership criterion and respect the family-wide vocabulary and composition rules.

1.3. Relationship to Verifiable Intent

This document is a vocabulary specification for the Verifiable Intent mandate format. It does not modify the Verifiable Intent specification itself. It declares a constraint-family namespace (environment.*) and the rules under which constraint types within that namespace operate, in the same manner that any extension of an existing vocabulary declares its own namespace and rules without modifying the host vocabulary’s grammar.

An open mandate, as used in this document and in [VI-CREDENTIAL-FORMAT], Section 4.5, is a Layer 2 mandate carrying constraints rather than final values, identified by a vct value with the .open suffix. environment.* constraints appear only in open mandates.

VI mandate validators that do not recognize the environment.* namespace MUST reject open mandates containing environment.* constraints, per [VI-CONSTRAINTS], Section 5.4: open mandates with unknown constraint types fail closed regardless of strictness mode, because an unevaluable constraint in an open mandate leaves agent authority unbounded. Because environment.* constraints appear only in open mandates, the rejection rule applies uniformly to every mandate in which environment.* constraints can appear.

Validators implementing this document MUST evaluate environment.* constraints in the order specified in Section 5 and MUST treat any failure as terminal for the mandate.

1.4. Document Organisation

Section 2 establishes terminology and references. Section 3 defines the constraint family and its membership criterion. Section 4 specifies family-wide vocabulary including the fields that appear, with identical semantics, on every member type, the field-scope taxonomy under which fields are classified, and the algorithm-agility framework under which signing-algorithm choice is made. Section 5 specifies family composition: conjunction semantics across family members, the L3 execution gate and its completeness rule, per-member diagnostic output, and the membership-criterion rationale these rules collectively encode. Section 6 specifies register discipline. Section 7 covers Security Considerations specific to family-wide concerns. Section 8 covers IANA Considerations including the proposed environment.* namespace registry. Appendix A documents the relationship between this specification and the two existing reference type specifications. Appendix B is reserved for future family-charter material per Section 6.7 Q3.

2. Terminology and References

2.1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document distinguishes between normative register and descriptive register, per the discipline catalogued at Section 6. Normative register uses RFC 2119 keywords in all capitals to bind implementers to specific behaviour. Descriptive register uses lowercase synonyms (or explicit non-keyword phrasing) when the prose explains intent, rationale, or context rather than imposing a requirement. The distinction is not stylistic; it determines what is conformance-checkable.

2.2. Terms

The following terms are defined for use in this document and in any specification of an environment.* constraint type that claims membership in the family.

Constraint family. A set of constraint types that share a common membership criterion, a common composition discipline, and a common family-wide vocabulary. The environment.* family is the family this document defines. The mandate.checkout.* and mandate.payment.* groupings within [VI-CONSTRAINTS] are not constraint families in the sense of this document; they are sub-namespaces of the transactional constraint vocabulary that share a prefix but do not share a membership criterion.

Membership criterion. A property that a constraint type MUST satisfy to qualify as a member of a constraint family. Membership in the environment.* family requires that the constraint type’s failure mode be gating: a failure of the constraint MUST terminate execution rather than degrade gracefully. Section 3 specifies the criterion in normative detail.

Transactional constraint. A constraint that describes a property of the transaction itself — what is being purchased, by whom, for how much, against which credential. The constraint families mandate.checkout.* and mandate.payment.* defined in [VI-CONSTRAINTS] are transactional. Transactional constraints are evaluated against fulfillment values derived from Layer 3 mandates. They do not require any external state observation at evaluation time.

Environmental constraint. A constraint that describes a property of the environment in which the transaction is executed — whether a venue is open, whether a wallet is funded, whether other relevant external conditions hold. Environmental constraints are evaluated against external state observations attested by an entity outside the transaction. The environment.* family is the set of environmental constraint types that satisfy the membership criterion of this document.

Attestation. A signed statement, issued by an entity outside the transaction, asserting that a specified environmental property held at a specified time. Attestations carry a binding to a subject (whose property is described), a binding to an issuer (whose key signed the statement), a freshness window (within which the attestation is acceptable), and an evaluation result (the property held or did not hold). The wire format and signing algorithm of an attestation are per-type concerns, specified by each constraint type. The role of the attestation in the verification path is family-wide and is specified by this document.

Issuer. The entity that produces an attestation by observing external state and signing the result. Each constraint type specification names the trust-root mechanism by which a verifier discovers an issuer's signing key (for example, [RFC7517] JWKS or [RFC8615] well-known key registry). The mandate-level binding to a specific issuer's signing key is enforced by the constraint instance, not by the family.

Mandate issuer. The party that constructs the Layer 2 mandate carrying environment.* constraints. The mandate issuer selects the constraint instances, populates per-instance fields (including the trust-root binding to a specific attestation issuer), and signs the mandate per [VI-CREDENTIAL-FORMAT]. The mandate issuer and the attestation issuer are distinct roles; an entity MAY occupy both roles in a deployment but the security analysis treats them separately.

Verifier. The party that evaluates an environment.* constraint at the moment of execution. A verifier fetches a fresh attestation per the constraint's binding, verifies the attestation signature against the issuer's published key, applies the family-wide vocabulary checks specified in Section 4 (including freshness against the constraint's max_attestation_age), and applies the per-type checks specified by the constraint type's specification. A verifier MUST NOT accept an agent's claim of constraint satisfaction in place of independent verification.

Agent. The autonomous principal that constructs Layer 3 mandates under the authority delegated by a Layer 2 mandate. In the family's evaluation path, an agent is a pre-Layer-3 verifier: it MUST itself satisfy every environment.* constraint before constructing Layer 3, with the agent-side discipline specified in Section 5.

Type specification author. The party that drafts and submits a constraint type specification proposing membership in the environment.* family. The type specification author is bound by the requirements of Section 3.4 and Section 4.2.2 governing how new types are specified.

Layer 2, Layer 3. Credential layers as defined in [VI-CREDENTIAL-FORMAT]. Layer 2 carries the user-signed mandate that includes constraint instances; Layer 3 carries the agent-signed fulfillment that the verifier accepts or rejects against the constraint instances. The terms are used in this document only to identify the points in the credential lifecycle at which family rules apply; this document does not modify the layer definitions.

Open mandate. A Layer 2 mandate carrying constraints rather than final values, identified by a vct value with the .open suffix per [VI-CREDENTIAL-FORMAT], Section 4.5. environment.* constraints appear only in open mandates. The unknown-constraint rejection rule of [VI-CONSTRAINTS], Section 5.4 applies uniformly to every mandate in which environment.* constraints can appear.

Family-wide field. A field whose semantics are identical across every constraint type in the family, independent of trust-root or evaluation mechanism. Family-wide fields are specified in this document.

Per-type field. A field whose semantics depend on the constraint type's trust-root mechanism or evaluation mechanism. Per-type fields are specified by each constraint type's specification under the field-scope taxonomy of Section 4.

2.3. References

Citations in this section are short forms; the canonical reference list is in Section 9.

2.3.1. Normative References

[RFC2119], [RFC8174], [RFC8126], [VI-CONSTRAINTS], [VI-CREDENTIAL-FORMAT].

2.3.2. Informative References

[RFC1918], [RFC7517], [RFC7646], [RFC7800], [RFC8615], [RFC8725], [RFC8914], [RFC9421], [RFC6982], [RFC-SD-JWT], [ENVIRONMENT-MARKET-STATE], [ENVIRONMENT-WALLET-STATE].

3. The environment.* Constraint Family

This section defines the environment.* constraint family. It specifies the membership criterion under which a constraint type qualifies as a member of the family, the namespace under which family members are named, and the discipline under which new family members may be added.

3.1. Family Definition

The environment.* constraint family is the set of Verifiable Intent constraint types that:

(1) declare a property of the environment in which a transaction is executed, rather than a property of the transaction itself;

(2) are evaluated against an attestation produced by an entity outside the transaction, rather than against fulfillment values derived from a Layer 3 mandate;

(3) gate execution unconditionally on the result of that attestation; and

(4) are named under the environment.* namespace.

Constraint types satisfying these four properties MAY claim membership in the family by publishing a constraint type specification that conforms to this document. Constraint types that satisfy fewer than all four properties are not members of the family, regardless of how their type identifier is named.

3.2. Membership Criterion

The membership criterion of the environment.* family is that the constraint type’s failure mode MUST be gating. A constraint type’s failure mode is gating if and only if every defined failure path of the type’s verification algorithm produces termination of execution rather than partial fulfillment, retry, or graceful degradation.

A constraint type whose failure mode is OR-tolerable — that is, a type whose failure can be survived by composition with other passing constraints, or whose failure leads to a permitted reduced-functionality execution path — is by definition outside this family. The reasoning is structural: a constraint that does not gate execution is not load-bearing for execution safety, and a constraint that is not load-bearing for execution safety does not require the family-wide vocabulary, composition discipline, or trust-root machinery this document specifies. Such a constraint may belong to some other family, or to no family at all, but it does not belong here.

The membership criterion is not a per-type design choice that the type specification author makes when proposing a new type. It is a property the type specification author MUST establish before proposing the type for the family. A proposed type whose failure mode is not demonstrably gating MUST be rejected from the family on this ground alone.

3.3. Namespace

Family members are named under the environment.* namespace, using the dot-notation pattern of [VI-CONSTRAINTS]. The first component of the type identifier is the literal string environment; the second component identifies the specific environmental property the type addresses; further components MAY be used by future revisions for sub-classification.

The environment.* namespace is registered for this purpose by Section 8 (IANA Considerations). Names within the namespace SHOULD be chosen to identify the environmental property addressed (for example, market_state, wallet_state, regulatory_status) rather than the implementation, the issuer, or the wire format.

IANA MUST NOT register a type identifier within the environment.* namespace that does not satisfy the membership criterion of Section 3.2. The namespace and the criterion are coupled by design: a registered type within environment.* is, by registration, a member of the family in good standing, and a verifier encountering an environment.* identifier MAY rely on the family's discipline holding for that type. Permitting the namespace to contain types that do not satisfy the criterion would defeat this property and is non-conforming.

3.4. Adding New Family Members

A new constraint type MAY be proposed for the environment.* family by a type specification author publishing a constraint type specification that:

(1) demonstrates that the proposed type satisfies the membership criterion of Section 3.2;

(2) specifies a trust-root mechanism (for example, [RFC7517] JWKS, [RFC8615] well-known key registry, or another mechanism with comparable security properties — meaning a mechanism that provides cryptographically authenticated key discovery, supports key rotation, and binds the discovered key to a specific issuer identity) under which a verifier discovers the issuer’s signing key;

(3) specifies a verification algorithm that consumes attestations produced under the trust-root mechanism and produces either constraint-satisfied or constraint-violated as terminal states, with no other terminal states;

(4) declares the per-type fields the type adds to the family-wide vocabulary, classifying each under the field-scope taxonomy of Section 4.2;

(5) declares the per-type MUST-implement signing algorithm and the per-type SHOULD/MAY extension set under the algorithm-agility framework of Section 4.3;

(6) declares the per-type distinguishing field used for per-member diagnostic disambiguation under the family composition discipline of Section 5.4, with a fallback specified for cases where the distinguishing field is not present in a failing instance; and

(7) conforms to the register discipline of Section 6 throughout.

These seven requirements correspond to the Expert Review criteria specified in Section 8.3 for IANA registry purposes. A constraint type specification meeting all seven requirements is conforming. A specification missing any requirement is not conforming and the proposed type MUST NOT be registered until the gap is closed.

The relationship among Section 3.1, Section 3.2, and this section is as follows. Section 3.1 specifies the four properties that characterise a family member. Section 3.2 names the gating-failure-mode property — property (3) of Section 3.1 — as the membership criterion. This section’s seven requirements establish the operational obligations a type specification must satisfy to

demonstrate the four properties: requirement (1) demonstrates property (3) of Section 3.1 directly; requirements (2) and (3) establish the operational basis for property (2) (attestation-driven evaluation); requirements (4) through (7) establish the family-wide vocabulary, composition, and register obligations a member type must respect. Property (1) of Section 3.1 (environmental, not transactional) and property (4) (named under the environment.* namespace) are pre-conditions that the type specification author asserts before invoking the seven requirements. Together, Sections 3.1, 3.2, and 3.4 specify the same family membership relation from descriptive, normative, and operational viewpoints respectively.

This document does not prescribe the working-group process under which a conforming specification is reviewed and registered; that process is the responsibility of the standards body owning the environment.* registry. This document specifies only the technical conditions a specification MUST meet to be eligible for registration.

3.5. Handling of Weakened Type Specifications

A registered family member’s specification may be revised over its lifetime. If a revision weakens the type’s failure mode below gating, the type’s specification no longer satisfies the membership criterion of Section 3.2, even though the type’s registration in the environment.* registry still asserts membership. This is a contradiction between registry state and specification content that this document does not resolve directly; resolution is a working-group concern, typically through deprecation per Section 8.4 followed by registration of a successor type.

Implementations MUST NOT silently accept the weakened semantics. If an implementation determines that a registered type’s current specification no longer satisfies the membership criterion, the implementation MUST reject mandates containing instances of that type and SHOULD report the contradiction to the working group rather than continue verification under the weakened specification. The MUST is the family’s fail-closed posture; the SHOULD is on discoverability of the contradiction by the registry’s governance body.

3.6. Relationship to Other Constraint Families

The environment.* family is one constraint family among potentially many in the Verifiable Intent ecosystem. The mandate.checkout.* and mandate.payment.* groupings defined by [VI-CONSTRAINTS] are sub-namespaces of the transactional constraint vocabulary; they are not constraint families in the sense of this document because they do not share a single membership criterion across their members. A future specification MAY define additional constraint families under the

same family-definition discipline this document establishes, with their own namespaces, membership criteria, and per-family vocabularies.

Composition of environment.* constraints with constraints from other families is specified in Section 5. The family relationship is unidirectional: environment.* constraints gate execution and are evaluated before transactional constraints; transactional constraints do not gate environment.* evaluation. This ordering is normative regardless of whether other families are defined in the future.

Constraint frameworks that bind agent authority to credential layers via Selective Disclosure JWT [RFC-SD-JWT] or HTTP Message Signatures [RFC9421], including Verifiable Intent's Layer 2 mandate format and similar agent-authorization protocols, are interoperable with the environment.* family without modification to either side. An environment.* constraint instance, embedded in the Layer 2 mandate's constraint list and covered by the mandate's JWS signature, inherits the mandate's [RFC7800] cnf-claim binding to the agent's proof-of-possession key. The verifier's evaluation path under Section 5.5 (environment.* before transactional constraints) sequences correctly with the host framework's own Layer-2-to-Layer-3 verification path: environment.* constraints gate the entire mandate evaluation, including any transactional constraints the host framework specifies. No registration of environment.* identifiers in the host framework's constraint registry is required for this interoperability; recognition of the environment.* namespace is sufficient for a host framework's verifier to delegate evaluation to an environment.*-aware verification path. Per [VI-CONSTRAINTS], Section 5.4 and Section 1.3 of this document, host frameworks whose validators encounter environment.* constraint instances without recognizing the namespace will reject the containing open mandates. This rejection is automatic; environment.*-aware verification paths operate against open mandates that the host-framework validator does recognize, without requiring host-framework registration of the environment.* namespace as a precondition.

4. Family-Wide Vocabulary

This section specifies the vocabulary that all environment.* constraint types share. Three classes of vocabulary are family-wide:

(1) fields whose semantics are identical across every member type, specified in Section 4.1;

(2) the field-scope taxonomy under which fields are classified as family-wide or per-type, specified in Section 4.2;

(3) the algorithm-agility framework under which signing-algorithm choice is made, specified in Section 4.3.

Per-type vocabulary — fields, signing algorithms, evaluation outputs, distinguishing identifiers — is specified by each constraint type's specification under the framework this section establishes. This section does not specify per-type vocabulary.

4.1. Family-Wide Fields

The following fields appear, with identical semantics, on every environment.* constraint instance.

4.1.1. attestation_url

Type: string (HTTPS URL). Required: Yes.

The HTTPS endpoint at which the verifier fetches the signed attestation for this constraint instance. The semantic role of attestation_url is identical across every environment.* constraint type: it names the location at which a fresh attestation can be obtained. Per-type variance in the wire protocol (HTTP method, request body shape, response envelope) is carried by per-type fields and per-type interface specifications under Section 4.2.3, not by attestation_url.

Verifiers MUST reject constraint instances whose attestation_url does not use the https scheme. This rejection is fail-closed: an attestation fetched over an unencrypted channel is by definition untrusted, and the constraint instance carrying such a URL is malformed regardless of whether the unencrypted endpoint would have returned a valid signature. Verifiers SHOULD apply additional transport-layer protections appropriate to the deployment context, including but not limited to rejection of URLs resolving to private address ranges, strict request timeouts, and bounded redirect handling. The specifics of these protections are deployment-defined.

The security implications of attestation_url — including server-side request forgery considerations and connection-time defences — are addressed in Section 7.7.

4.1.2. max_attestation_age

Type: integer (seconds). Required: Yes.

The maximum age in seconds, measured from the attestation's signing time to the verifier's evaluation time, beyond which the attestation MUST NOT be accepted. The mandate issuer declares this value at

constraint-instance construction time. Verifiers MUST reject attestations whose age exceeds max_attestation_age, regardless of whether the attestation's own expiry has not yet passed.

max_attestation_age MUST be a positive integer. A constraint instance lacking an explicit max_attestation_age is malformed and verifiers MUST reject it. There is no default value at the family layer; specifications adding default values per constraint type are non-conforming because the freshness window is the family's primary security property, and a default at any layer recreates the implicit-freshness exposure the family is designed to close.

The freshness window is the primary exploitable surface of any environment.* constraint. The window is the period during which a signed attestation, valid at the moment of signing, may no longer reflect the current state of the property the constraint addresses. An attestation that is technically still inside its issuer's TTL but is operationally stale — that is, its signing time predates a relevant change in the underlying environmental property — is the canonical failure case: market state can change between signing and execution, wallet state can change between signing and execution, and any environmental property the family addresses can change between signing and execution. The mandate issuer is the party best positioned to declare the maximum tolerable window for the gated decision; max_attestation_age is the field through which that declaration is made.

The attestation issuer's TTL (the issuer-side time after which the issuer no longer claims the attestation is valid) and max_attestation_age (the consumer-side time after which the verifier no longer accepts the attestation) serve different parties and MUST be independently configurable. A verifier MUST honour both: an attestation that has passed either bound is unacceptable. The narrower of the two governs in practice.

The security implications of max_attestation_age — including attestation replay considerations — are addressed in Section 7.1.

4.2. Field-Scope Taxonomy

The environment.* constraint family contains fields that appear under identical names in multiple constraint types. A shared field name does not, by itself, imply shared semantics: some fields carry identical meaning across types, while others have parallel-but-mechanism-specific meaning tied to each type's trust-root or evaluation mechanism. To prevent ambiguity, each field in the family MUST be classified by its specification author under one of the categories defined below.

The family currently recognises three scope categories:

family-wide-trust-root-agnostic. The field carries identical semantics across every environment.* constraint type, independent of the trust-root mechanism each type uses. A single verification code path handles the field for every type in the family.

per-type-trust-root-mechanism-bound. The field's operational semantics depend on the specific trust-root mechanism of the constraint type ([RFC7517] JWKS, [RFC8615] well-known key registry, or another mechanism). The field name may appear in multiple specifications with parallel but mechanism-specific semantics.

per-type-evaluation-mechanism-bound. The field's operational semantics depend on the constraint type's evaluation mechanism: the output shape the evaluator produces or the wire-protocol binding to the evaluator. The field name may appear in multiple specifications with parallel but evaluation-specific semantics; distinct from per-type-trust-root-mechanism-bound in that the binding is to how the attestation is produced and shaped, not to how the signing key is discovered.

Within Section 4.2, the term "field" encompasses both constraint schema fields and, where operationally relevant, claims within the signed attestation payload.

4.2.1. Scope Declarations for Family-Wide Fields

The family-wide fields specified in Section 4.1 are classified as follows:

attestation_url: family-wide-trust-root-agnostic. Its semantic role — the HTTPS endpoint at which the verifier fetches the signed attestation — is identical across every environment.* constraint type. Per-type variance in HTTP method or request body is carried by neighbouring per-type fields, not by the URL field itself.

max_attestation_age: family-wide-trust-root-agnostic. The freshness window is a temporal property of the attestation signing event, independent of how the verifier retrieves the signing key.

4.2.2. Scope Declarations for Per-Type Fields

Type specification authors MUST declare the scope of every per-type field they introduce. Declarations MUST appear in a Field Scope Declaration section of the type specification. The declaration MUST classify the field under one of the three categories of Section 4.2 and MUST justify the classification in prose.

The type specifications referenced in Appendix A carry such declarations for the per-type fields they introduce. This document does not enumerate those fields, because doing so would couple the family-definition layer to the current set of registered types in a way that future revisions of either type's specification could invalidate.

4.2.3. Adding Scope Categories

The three categories of Section 4.2 are sufficient for every field introduced by every member type registered at the time of this document's publication. Type specification authors proposing new fields SHOULD place them under one of the three existing categories.

A type specification author MAY propose a new scope category. Such a proposal is a working-group revision of this document, not a per-type design choice. The proposing specification MUST demonstrate that no existing category is sufficient for the field's classification, and the working group MUST converge on the new category's definition and integration with the existing three before any registered type uses it.

4.3. Algorithm Agility

The environment.* constraint family is algorithm-agnostic at the family layer. This document does not name a signing algorithm that all family members must use, and it does not prefer any algorithm over any other. Algorithm choice is a per-type concern, made by each constraint type specification under the framework this section establishes.

The framework follows the discipline of [RFC8725], Section 3.1: each constraint type specification MUST declare a single MUST-implement signing algorithm that all conformant verifiers for that type MUST support, and MAY declare a SHOULD-implement extension set and a MAY-implement extension set. Verifiers negotiate the algorithm per constraint instance by reading the attestation's algorithm declaration and confirming that the declared algorithm is in their supported set for the constraint type. Verifiers MUST reject attestations whose algorithm is not in the type's supported set; this rejection is fail-closed and is not a permitted silent downgrade.

Algorithm deprecation within a registered family member is a per-type concern. Type specification authors SHOULD specify, in the type's specification, a deprecation mechanism for the type's MUST-implement algorithm before that algorithm is needed — including the conditions under which the algorithm is deprecated, the timeline for verifier migration to a successor MUST-implement, and the backward-

compatibility guarantees during the transition. The family-wide obligation is structural: every type specification MUST declare its current MUST-implement algorithm, and verifiers MUST honour the declaration without fallback. Deprecation discipline ensures that the family does not inherit a brittle commitment to any specific algorithm beyond the cryptographic lifetime that algorithm provides.

The framework is deliberately minimal at the family layer because the security properties of signing-algorithm choice are well-specified by the JOSE family of specifications and by [RFC8725]. The family-layer obligation is structural: every type specification MUST declare its choice explicitly, and verifiers MUST honour the declaration without fallback. This obligation prevents the family from accidentally inheriting a single-algorithm constraint by implementation convergence rather than by specification.

A type specification's choice of MUST-implement algorithm is not constrained by this document. The choice is properly made on the basis of the reference implementation's signing stack, the wider VI ecosystem's existing JWS infrastructure, the operational properties of the algorithm at the type's expected verification volume, and the security properties relevant to the type's threat model. This document is silent on these factors because they are per-type concerns; it requires only that the choice be made, declared, and respected.

5. Family Composition

This section specifies how environment.* constraints compose with each other and with constraints from other families. Six concerns are addressed: conjunction semantics across family members (Section 5.1), mixed pass/fail handling (Section 5.2), the L3 execution gate and its completeness rule (Section 5.3), per-member diagnostic output (Section 5.4), composition with other families (Section 5.5), and the rationale these rules collectively encode (Section 5.6).

5.1. Conjunction Semantics

The environment.* family is a conjunction. A mandate satisfies the family's gate if and only if every environment.* constraint instance in the mandate passes. There is no partial-fulfillment path within the family, no quorum across members, and no precedence by which one member can override another.

This semantic is family-wide and is not a per-type design choice. A constraint type whose composition with sibling family members is anything other than conjunction is by definition outside the family,

because the membership criterion of Section 3.2 requires that each type's failure mode be gating, and a gating failure cannot be survived by composition with passing siblings.

5.2. Mixed Pass/Fail

When one environment.* constraint instance passes and another fails in the same mandate, the family's gate fails. The passing instance does not rescue the failing instance. This rule applies regardless of:

(1) whether the instances are of the same type or different types;

(2) whether the instances bind to the same subject or to different subjects;

(3) the order in which the instances appear in the mandate; and

(4) the order in which the verifier evaluated the instances.

A verifier that returns a passing result for a mandate containing one passing and one failing environment.* constraint is non-conforming.

5.3. L3 Execution Gate and Completeness

The environment.* family gates Layer 3 acceptance. A verifier MUST NOT accept a Layer 3 mandate whose Layer 2 mandate carries any failed environment.* constraint, regardless of whether the verifier evaluated the failed constraint first, last, or in any other order.

The verifier MUST evaluate every environment.* constraint in the mandate to completion before refusing Layer 3, even after the first failure has been observed. This is the completeness requirement. Short-circuit evaluation of environment.* constraints — terminating the family-evaluation pass at the first failure without evaluating subsequent members — is non-conforming because it denies downstream consumers the per-member evidence the family's diagnostic output (Section 5.4) depends on.

The completeness requirement applies to the verifier-side L3-acceptance evaluation path. It does not apply to the agent-side pre-L3-creation evaluation path. An agent constructing Layer 3 MUST itself satisfy every environment.* constraint before constructing Layer 3, and an agent that observes any environment.* failure during pre-L3-creation evaluation MUST halt construction at the first failure as a fail-closed agent-side response. The agent's halt is operationally distinct from the verifier's diagnostic-completeness obligation: the agent's purpose is to avoid producing a Layer 3 that the verifier would refuse; the verifier's purpose is to produce complete diagnostic output even when refusing.

Evaluation of constraints from other families (for example, transactional constraints under mandate.*) after a confirmed environment.* failure remains implementation-defined. A verifier MAY evaluate them for diagnostic completeness or MAY skip them for efficiency, provided the resulting refusal of Layer 3 is unambiguous.

5.4. Per-Member Diagnostic Output

Each failed environment.* constraint MUST produce its own entry in the verifier's violations output. Verifiers MUST NOT collapse multiple environment.* failures into a single generic violation entry.

Each violation entry MUST carry two identifiers:

(1) the array index of the failing constraint within the mandate's constraints array, as the primary machine identifier. The index is unambiguous across any combination of constraint types and across mandates containing multiple instances of the same type;

(2) a human-readable identifier derived from the failing constraint's distinguishing field, as the secondary identifier intended for operator diagnostics and dispute resolution.

The selection of the distinguishing field is a per-type obligation. Each constraint type specification author MUST declare which field of the type is the distinguishing field, and MUST specify a fallback when the distinguishing field is not present in the failing instance (for example, when the constraint's signed attestation could not be obtained at all). The reference type specifications cited in Appendix A carry such declarations for their respective types.

The diagnostic output requirement is family-wide because the family's audit value depends on it. A mandate carrying multiple environment.* constraints — whether multiple instances of the same type or instances of multiple types — produces, at refusal, an audit record showing exactly which environmental conditions failed and which held. Collapsing per-member diagnostics destroys that record.

5.5. Composition with Other Families

environment.* constraints MUST be evaluated before constraints from other families in the same mandate. If any environment.* constraint fails, the verifier MUST refuse Layer 3 regardless of how constraints from other families would evaluate. The ordering is one-directional: environment.* gates execution, and other families do not gate environment.* evaluation.

The rationale is structural. Evaluating transactional constraints (allowed merchants, allowed payees, amount ranges) before confirming the execution environment is valid exposes the verifier to a time-of-check-to-time-of-use race in which a transactional constraint passes against a Layer 3 that names an environment the verifier has not yet attested. environment.* evaluation precedes transactional evaluation so that the environment is established before the transaction is authorised against it.

This ordering is normative regardless of whether other constraint families exist at the time of this document's publication or are added later. A future family with its own membership criterion and its own family-definition specification MUST document its own ordering relationship to environment.*, but environment.* itself is not subject to ordering preemption by any other family.

5.6. Rationale

The composition rules of Sections 5.1 through 5.5 are not arbitrary. Two arguments, co-equal and reinforcing, motivate them.

The semantic argument: each member of the environment.* family answers an independent question about the world at the moment of execution — is this venue open? is this wallet funded? does this counterparty's regulatory status hold? Because the questions are independent, their answers compose as conjunction, not disjunction: a failure on any member removes the basis for execution. There is no construction in which "the venue is open OR the wallet is funded" is the correct gate for an autonomous transaction. Both must hold.

The architectural argument: conjunction also falls out of the family's membership criterion (Section 3.2) directly. A constraint type whose failure mode is OR-tolerable is, by the criterion, outside the family. If a constraint's failure is survivable by composition with passing siblings, the constraint is not gating execution; if it is not gating execution, it does not satisfy the membership criterion; if it does not satisfy the criterion, it is not in the family. Conjunction is therefore not a design choice the working group makes when registering each new type — it is a consequence of the membership criterion the type satisfied to enter the family in the first place.

The two arguments meet at the same conclusion from independent directions. The semantic argument applies regardless of how the family is defined. The architectural argument applies regardless of what the family means. That both arguments converge on conjunction is the strongest evidence that conjunction is the right composition rule for environment.*.

The per-member diagnostic requirement of Section 5.4 follows from the same logic. Collapsing per-member diagnostics would destroy the audit trail that makes the family load-bearing for downstream dispute resolution. Preserving per-member diagnostics makes every signed attestation recoverable from the verifier's output, and makes dispute resolution an application-layer concern with complete evidence rather than a debugging exercise. Both type specifications cited in Appendix A already encode this discipline; this document promotes it from per-type rationale to family-wide normative requirement so that every future family member inherits it without re-litigation.

6. Register Discipline

This section specifies the discipline under which RFC 2119 keywords are used in this document and in any specification of an environment.* constraint type that claims membership in the family. The discipline distinguishes normative register from descriptive register and constrains the use of each.

The discipline is family-wide. A constraint type specification that does not conform to this section is non-conforming as a family member, regardless of the soundness of its technical content, because mixed register defeats the conformance-checkability that makes the family's normative statements actionable for implementers.

6.1. Normative Register

Normative register uses RFC 2119 keywords (MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, OPTIONAL) in all capitals, per [RFC2119] and [RFC8174], to bind implementers to specific behaviour.

Normative register specifies behaviour that an implementation either satisfies or fails to satisfy; it is conformance-checkable. Any RFC 2119 keyword in all capitals MUST appear only in a sentence that imposes a requirement on a named implementer (an agent, a verifier, a mandate issuer, an attestation issuer, a type specification author, IANA, or a designated expert). The Section 2.1 boilerplate per [RFC8174] binds the interpretation throughout this document.

6.2. Descriptive Register

Descriptive register explains intent, rationale, precedent, context, or structural reasoning. It uses lowercase synonyms (or explicit non-keyword phrasing) where RFC 2119 keywords would otherwise appear.

Descriptive register is not conformance-checkable. It exists to make the normative register actionable: a reader of normative requirements is better served by descriptive prose explaining why the requirement holds than by the requirement standing alone without context. Descriptive prose SHOULD use lowercase synonyms ("must" rather than MUST, "should" rather than SHOULD, "may" rather than MAY) when no new requirement is being imposed. Where the lowercase synonym would be ambiguous about whether it imposes a requirement, explicit non-keyword phrasing ("the section establishes", "the type specification author’s choice is", "the framework requires") is preferred over either capitalisation.

The discipline matters because a specification that mixes normative and descriptive register — for example, by using all-capitals MUST in a rationale paragraph that does not impose a requirement — produces ambiguous conformance: a reviewer or implementer reading prose with all-capitals keywords in descriptive context cannot tell whether the text imposes a requirement or merely describes one. Resolving such ambiguity requires reading the surrounding sentences to infer intent, which is precisely the work the discipline is designed to avoid.

6.3. Redundancy Avoidance

[RFC2119], Section 3 establishes that SHOULD and RECOMMENDED carry identical normative force; [RFC2119], Section 5 establishes that MAY and OPTIONAL carry identical normative force. Field labels, section headers, and parenthetical qualifiers that duplicate the normative keyword present in the containing sentence add visual weight without adding normative content.

A specification SHOULD NOT pair a label such as "(RECOMMENDED)" with a sentence whose verb is already SHOULD, and SHOULD NOT pair a label such as "(OPTIONAL)" with a sentence whose verb is already MAY. The keyword in the containing sentence carries the requirement; the parenthetical label adds register noise.

This rule applies to specification authors as a stylistic discipline. A specification that violates it remains technically conformant; the rule’s purpose is to keep the document surface clean enough that the discipline of Sections 6.1 and 6.2 is visible to a reviewer scanning for register issues.

6.4. Scope of Application

Normative requirements MUST name the implementer they bind. A requirement that does not name its implementer is ambiguous and the ambiguity is not resolved by context.

Where a requirement could be read at multiple layers of the stack — for example, "verifiers MUST re-verify" read as a network-transport obligation versus an application-layer obligation — the specification text MUST name the application-layer scope explicitly. A sentence such as "verifiers MUST re-verify the attestation against the chain’s then-current state at settlement time" reads ambiguously without scope: it could bind a chain client, an RPC layer, a cryptographic library, or the VI verifier’s L3-acceptance logic. The specification text MUST name which.

The application-layer scope of family-wide requirements in this document is the VI verifier’s L3-acceptance logic. Requirements at other layers (network transport, cryptographic library, chain client) are out of this document’s scope and are governed by the specifications appropriate to those layers.

6.5. Forward Rule for New Sections and New Specifications

New sections introduced in future revisions of this document, and all sections of any constraint type specification that claims membership in the environment.* family, MUST apply the rules of Sections 6.1 through 6.4 at introduction.

Editorial register polish — conversion of all-capitals to lowercase in descriptive context, removal of RECOMMENDED/SHOULD redundancy, application-layer scope naming — is a patch-level revision and does not require a minor-version bump in any specification that adopts patch-level versioning. The rule is not that the discipline be applied perfectly on first draft, but that it be applied without exception by the time a specification is presented for working-group adoption.

6.6. Relationship to Section 2.1 Conventions

This section extends the Section 2.1 Conventions and Definitions statement. Section 2.1 binds the interpretation of RFC 2119 keywords when they appear in all capitals per [RFC8174]. This Section 6 specifies the editorial discipline for when RFC 2119 keywords SHOULD appear in all capitals (Section 6.1, normative register) versus when they SHOULD NOT (Section 6.2, descriptive register). Together, Section 2.1 and Section 6 establish the register discipline of the environment.* family.

6.7. Open Questions

This section reproduces three questions that the type specifications cited in Appendix A raise about register discipline and that the working group has not yet converged on. The questions are reproduced here so that working-group review of this document addresses them as family-wide concerns rather than per-type concerns. The current draft takes provisional positions on each; the working group is invited to revise.

Q1. Is this section informative — describing the discipline that the family's normative sections already follow — or normative — binding future type specifications to the rules at Sections 6.1 through 6.5? Provisional position: Sections 6.1 through 6.5 are treated as normative (with all-capitals RFC 2119 keywords throughout) and Sections 6.6 and 6.7 as informative. The working group may reclassify any subsection.

Q2. Should the family adopt a formal register-discipline audit cadence — for example, a pre-release audit before each minor version of a type specification — or is the ad-hoc audit pattern established

by the type specifications cited in Appendix A sufficient? Provisional position: The current draft does not mandate a cadence; the rule of Section 6.5 is that the discipline be applied by the time a specification is presented for adoption.

Q3. What is the relationship between this section and other family-charter material that may be added to future revisions? Provider neutrality, peer-author coordination discipline, patch-level versioning conventions, and lockstep commit patterns across sibling specifications are family-charter concerns that the type specifications cited in Appendix A have raised. Provisional position: The working group decides whether such material consolidates into a future revision of this section, into a separate family-charter section, or into a hybrid structure; the current draft does not foreclose.

7. Security Considerations

This section enumerates security concerns that apply family-wide to environment.* constraints. Per-type security considerations — those tied to a specific trust-root mechanism, a specific signing algorithm, or a specific evaluation mechanism — are addressed by each constraint type's specification. The family-wide concerns below apply to every member type and to every implementation of every member type.

7.1. Attestation Replay

A signed attestation is a bearer artifact within its acceptance window. Any party in possession of a valid signed attestation may present it to a verifier, and the verifier — applying only the cryptographic checks the attestation supports — will accept it. The acceptance window is bounded by the attestation's expiry and further narrowed by the constraint instance's max_attestation_age (Section 4.1.2), but within that window the attestation is replayable.

For deployment contexts where attestation reuse would authorise duplicate state changes (for example, payment execution, where an attestation justifying one settlement could otherwise justify a second settlement against the same wallet within the freshness window), verifiers MUST maintain a per-attestation deduplication cache keyed on the attestation's unique identifier. For deployment contexts where attestation reuse is benign (for example, idempotent read-only operations), verifiers SHOULD maintain a per-attestation deduplication cache as defence in depth, but MAY rely on the freshness window alone. The unique identifier and the cache TTL are per-type concerns: each constraint type specification names the field

its attestations carry as the unique identifier, and the cache TTL is bounded by max_attestation_age plus an implementation-defined margin to absorb clock skew.

The replay surface is family-wide because it follows from the bearer-artifact property of signed attestations, which is itself family-wide: every environment.* attestation is a signed statement that a verifier accepts on the strength of the signature, and any signed statement that travels over the wire is replayable to any verifier that accepts the same signature. The specific mitigation varies by deployment context.

7.2. Issuer Trust Bootstrapping

The security of any environment.* constraint depends on the verifier correctly establishing the attestation issuer's signing key. Each constraint type names a trust-root mechanism (Section 4.2) under which a verifier discovers the key — JWKS at a known HTTPS URL, a well-known key registry per [RFC8615], or another mechanism — but every trust-root mechanism reduces, eventually, to an out-of-band initial trust establishment.

Verifiers SHOULD establish initial trust in an attestation issuer through at least one channel independent of the trust-root mechanism: DNSSEC for the issuer's domain, certificate transparency log inspection for the issuer's TLS certificates, published key fingerprints in a venue independent of the issuer's primary infrastructure, or out-of-band fingerprint comparison with a known-good party. The specifics of initial-trust channels are deployment-defined; the family-wide requirement is that no implementation rely on the trust-root mechanism alone to bootstrap trust in a previously-unknown issuer.

This concern is family-wide because the trust-root mechanism's job is to make ongoing key discovery secure once initial trust is established, not to establish initial trust ex nihilo. A verifier that fetches an unknown issuer's JWKS over HTTPS and accepts whatever keys are returned has authenticated only that the JWKS was served by the host named in the URL. It has not authenticated that the host named in the URL is the issuer the mandate intended. Initial trust is the gap between those two facts.

7.3. Constraint Stripping and Insertion

An attacker with the ability to modify a Layer 2 mandate could remove environment.* constraint instances from the mandate before the agent or verifier evaluates it. A mandate with environment.* constraints stripped expands the agent's authority to environments the user did not authorise.

The defence against constraint stripping is the Layer 2 signature, which covers the full constraint list as presented at issuance time. Any modification to the constraint list — including removal or insertion of an unauthorised constraint — invalidates the signature. Verifiers MUST reject any Layer 2 mandate whose signature does not validate over the full constraint list as presented, and implementations MUST NOT accept mandates whose verified signature covers only a subset of the declared constraints.

The same defence applies to constraint insertion: a mandate issuer acting outside the user's authorisation cannot add a constraint instance to a user-signed mandate without invalidating the user's signature. Note that this defence depends on the mandate issuer's honesty regarding what the user authorised at the time the mandate was signed; it does not protect against a mandate issuer who constructs a mandate with constraint instances the user did not authorise. Trust in the mandate issuer's faithfulness to user intent is a deployment-level concern outside the scope of this document.

This concern is family-wide because it follows from the family's insertion into Verifiable Intent's Layer 2 mandate format: any constraint in any family is subject to stripping and insertion, and the Layer 2 signature is the family-agnostic defence. The family inherits the defence; it does not add new defences.

7.4. Substitution of Attestation Endpoints

A more subtle attack than stripping is substitution: an attacker who can modify a Layer 2 mandate replaces a constraint's attestation_url with a URL pointing to an attacker-controlled endpoint that returns valid-looking signed responses. The signature on the substituted attestation is, by construction, valid against whatever key the attacker chose; the trust-root mechanism's binding to a specific issuer's key is what defeats the attack.

The defence is per-type, but its family-wide minimum is specified here. Each constraint type specification MUST identify, at minimum, two binding fields signed into the Layer 2 mandate alongside attestation_url: an issuer-identifier binding (the field that names which attestation issuer's key is acceptable) and a key-identifier

binding (the field that names which specific key, within that issuer's key set, is acceptable). For [RFC7517] JWKS-based types, this typically means a JWKS URL, a key identifier, and an issuer identifier; for [RFC8615] well-known key registry-based types, this typically means a key identifier together with the issuer identity from which the key registry URL is derived. An attacker who substitutes attestation_url MUST also substitute every binding field to evade detection, and the substitution is detectable provided the verifier honours every binding.

The family-wide minimum is that every constraint type's binding fields — at minimum, the issuer-identifier binding and the key-identifier binding — MUST be signed into the Layer 2 mandate by the mandate issuer at issuance time and MUST be honoured by verifiers at evaluation time. A type specification whose verification algorithm does not honour every declared binding field, or whose declared binding fields do not satisfy the family-wide minimum, is non-conforming. This minimum applies independent of trust-root mechanism: future trust-root mechanisms admitted under Section 3.4(2) ("comparable security properties") inherit the same issuer-identifier and key-identifier binding obligations.

7.5. Mandate-Issuer-Versus-Attestation-Issuer Trust Separation

Section 2.2 distinguishes the mandate issuer (the party that constructs the Layer 2 mandate) from the attestation issuer (the party that signs the attestation the verifier evaluates against). The two roles are separately trusted and separately compromised. A compromised mandate issuer could construct mandates with attestation bindings pointing to compromised attestation issuers; a compromised attestation issuer could sign valid attestations for false world-states. Each compromise is independently consequential, and each role's trust establishment is independently the deployment's responsibility.

The family does not specify formal trust-establishment procedures for either role; that work belongs in Verifiable Intent's threat model and in deployment-specific operational documentation. The family-wide requirement is that the two roles MUST NOT be conflated by an implementation. An implementation that treats a mandate issuer's signature as evidence about the attestation issuer's signing key — or vice versa — is non-conforming because the two signatures bind different facts and a verifier acting on one as evidence for the other has reduced its security to whichever role is currently more trusted.

A concrete check that follows from this requirement: a verifier MUST NOT use the mandate issuer's identity (or any binding field attesting to the mandate issuer) as input to the verification of the attestation's signature. The attestation signature is verified against the attestation issuer's key, discovered through the trust-root mechanism, with no mandate-issuer-derived inputs. This check is conformance-testable.

This concern is family-wide because the role distinction is family-wide: every environment.* constraint instance carries both a mandate-issuer signature (over the Layer 2 mandate) and an attestation-issuer signature (over the attestation the constraint binds to), and every verification path involves both signatures. Conflating them is a verification-path error rather than a per-type error.

7.6. Time Synchronisation

The freshness check of Section 4.1.2 (max_attestation_age) requires that the verifier's clock is reasonably synchronised with the attestation issuer's clock. A clock skew exceeding max_attestation_age would cause the verifier to reject all attestations from a correctly-operating issuer.

Implementations SHOULD use clock-synchronisation mechanisms appropriate to the deployment context (such as NTP) and SHOULD log clock-skew-related verification failures distinctly from other failure modes to aid diagnosis. The family does not specify a clock-synchronisation mechanism; the family-wide requirement is that implementations use a mechanism whose accuracy is bounded below the smallest max_attestation_age value the implementation expects to encounter in production.

This concern is family-wide because max_attestation_age is family-wide. A constraint type specification that uses a clock-anchored field other than max_attestation_age inherits the same concern; this section's requirement applies to whichever clock-anchored field the type uses.

7.7. SSRF and Other Verifier-Initiated Fetch Risks

The attestation_url field of every environment.* constraint instance is a mandate-controlled URL that the verifier will fetch. Without protection, this is a server-side request forgery primitive: an attacker who can modify the Layer 2 mandate (or who can introduce constraint instances at mandate construction time) can direct the verifier to fetch arbitrary URLs.

Verifiers MUST apply transport-layer protections appropriate to the deployment context. Family-wide minimum requirements:

(1) reject URLs that do not use the https scheme (already required by Section 4.1.1);

(2) reject URLs whose hostnames resolve to address ranges not appropriate for outbound verifier traffic. At minimum, this includes the [RFC1918] private address ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16), loopback (127.0.0.0/8), link-local (169.254.0.0/16), and any deployment-specific exclusion list;

(3) bound request timeouts to a value short enough that verifier-side SSRF attempts cannot extract significant information through timing observation;

(4) bound redirect handling to at most one redirect, and apply the same address-range and scheme checks to redirect targets;

(5) re-resolve the hostname at connection time and apply the address-range checks of (2) to the resolved IP, defending against DNS rebinding attacks where a domain resolves to a public IP at fetch-time and a private IP at connection-time.

The specific timeout values, deployment-specific exclusion list extensions, and redirect-handling specifics beyond the minima above are deployment-defined. Per-type specifications MAY tighten these requirements but MUST NOT relax them; a type specification that permits non-HTTPS URLs, omits any of the address-range exclusions in (2), or admits unbounded redirects is non-conforming.

This concern is family-wide because every environment.* constraint instance carries an attestation_url and every verifier fetches it. The mitigation is family-wide minimum plus deployment-specific tightening.

7.8. Signer Failure Handling

The trust-root mechanisms specified by per-type specifications (e.g., [RFC7517] JWKS, [RFC8615] well-known key registry) provide cryptographically authenticated key discovery under correct operation. They do not address the failure mode in which an attestation issuer publishes signatures or key material that, while cryptographically well-formed, are operationally incorrect — for example, signatures over stale or misconfigured state, or key-rotation events that publish keys in an inconsistent intermediate state. Such failures are not theoretical: cryptographic infrastructure depending on similar trust-root mechanisms (DNSSEC

zone-signing key rollovers, certificate transparency log misissuance) has experienced documented outages of this class.

A constraint type's verification algorithm under Section 3.4(3) terminates as constraint-satisfied or constraint-violated; it does not have a third terminal state for "issuer is currently malfunctioning." Verifiers SHOULD therefore implement an application-layer mechanism analogous to the Negative Trust Anchor pattern of [RFC7646] under which a verifier MAY temporarily decline to validate against a specific issuer's keys when that issuer is known to be publishing incorrect attestations. Verifiers SHOULD additionally surface the reason for declined verification through a mechanism analogous to Extended DNS Errors [RFC8914], so that downstream consumers (agents, mandate issuers, audit systems) can distinguish issuer-failure refusals from constraint-violation refusals.

The family-wide requirement is operational rather than cryptographic: the verification algorithm continues to fail closed regardless of whether issuer-failure handling is implemented, and an agent observing any failure halts construction of Layer 3 per Section 5.3. The Negative Trust Anchor and Extended DNS Errors patterns do not weaken the failure-closed posture; they make the cause of refusal diagnosable by humans and audit systems. Implementations choosing not to implement these patterns inherit a coarser failure mode in which all verifications against a malfunctioning issuer fail without distinguishing cause, which is conformant but operationally degraded.

This concern is family-wide because every environment.* constraint type relies on a trust-root mechanism whose failure modes are external to the cryptographic verification path, and the operational consequences of those failures (mass agent halt, indistinguishable refusal causes) are family-wide regardless of which trust-root mechanism the type uses.

8. IANA Considerations

This document requests that IANA establish a registry for environment.* constraint type identifiers and register this document as the family-definition document for the namespace. The registry's scope, registration policy, and expert review criteria are specified below.

The Verifiable Intent specification [VI-CONSTRAINTS] establishes a Constraint Type Registry for VI mandate constraint types. The environment.* registry described here is either a sub-registry of that VI Constraint Type Registry or a separate registry coordinated with it; the working group's choice between these two structures does not affect the registration content this document specifies and is left to the working group at adoption time.

8.1. The environment.* Namespace

IANA is requested to register the environment.* namespace as a constraint type prefix in the registry of Section 8.2. The namespace is the literal string environment followed by a dot followed by one or more dot-separated components identifying a specific environmental property and, optionally, sub-classifications.

This document is registered as the family-definition document for the environment.* namespace. Specifications proposing new environment.* type identifiers MUST conform to this document. IANA MUST NOT register a specification that does not conform, regardless of the technical merit of the proposed type.

8.2. The environment.* Constraint Type Registry

IANA is requested to establish a registry titled "Verifiable Intent environment.* Constraint Types" (or a name the working group prefers) with the following structure.

Registry fields. Each registration entry has the following fields:

* Type identifier — the dot-separated string under the environment.* namespace (for example, environment.market_state, environment.wallet_state).

* Defined In — a reference to the constraint type specification document that defines the type. The reference SHOULD be a stable identifier (RFC number, Internet-Draft name and revision, or a permanent published-document URL).

* Version — the version of the type specification at the time of registration.

* Status — one of: Provisional, Stable, Deprecated.

* Distinguishing Field — the field per Section 5.4 that the type's diagnostic output uses for per-member identification. The fallback for cases where the distinguishing field is not present in a failing instance MUST be either listed here or referenced from the Defined-In document.

* MUST-Implement Algorithm — the per-type signing algorithm declared per Section 4.3.

* Notes — implementation-status references per [RFC6982], known-issues references, or coordination references with sibling specifications. May be empty.

Initial registry contents. This document does not register any specific type identifier. The two reference type specifications cited in Appendix A are expected to register environment.market_state and environment.wallet_state respectively under their own working-group-adoption procedures. Registration of those types is the responsibility of their respective specifications, not of this document.

Registration policy. Registration follows the Specification Required policy of [RFC8126], Section 4.6 with Expert Review by designated experts the working group appoints. The criteria the experts apply are specified in Section 8.3.

8.3. Expert Review Criteria

Designated experts evaluating a proposed registration MUST verify that the proposed type satisfies all seven requirements of Section 3.4:

(1) the type satisfies the membership criterion of Section 3.2 (failure mode is gating);

(2) the type's specification names a trust-root mechanism with comparable security properties as defined in Section 3.4(2);

(3) the type's verification algorithm produces only constraint-satisfied or constraint-violated as terminal states;

(4) the type's per-type fields are classified under the field-scope taxonomy of Section 4.2;

(5) the type's per-type MUST-implement signing algorithm and extension sets are declared per Section 4.3;

(6) the type's per-type distinguishing field for diagnostic disambiguation is declared per Section 5.4, with a fallback specified;

(7) the type specification conforms to the register discipline of Section 6.

A proposed registration that fails any of these checks MUST be rejected by the designated experts. Rejection is not punitive; it indicates that the gap should be closed in the proposed specification before the registration is reconsidered. Designated experts SHOULD provide specific feedback identifying which check the proposed registration failed and what closure of the gap would look like.

Designated experts SHOULD also evaluate proposed registrations for:

* Namespace coherence. The proposed identifier SHOULD identify the environmental property the type addresses (for example, market_state, wallet_state, regulatory_status), not the implementation, the issuer, or the wire format. An identifier such as environment.fooissuer_jwt is poorly chosen even if its specification is technically conformant.

* Coordination with sibling types. Where a proposed type's vocabulary overlaps with a registered type's vocabulary, the proposed specification SHOULD use the existing field name with the existing semantics, or SHOULD justify why a divergent name or semantics is necessary. Proliferation of near-synonymous fields across types defeats the family's vocabulary discipline.

* Implementation status. Proposed registrations from specifications that document at least one running implementation per [RFC6982] SHOULD be preferred over proposed registrations from specifications without running implementations, because the family's discipline depends on every member being implementable end-to-end with the family's vocabulary.

These additional criteria are guidance for designated experts. They are not normative requirements that proposed registrations MUST satisfy.

8.4. Status Field Lifecycle

The Status field of Section 8.2 has three values:

Provisional. The type’s specification has been adopted by the working group but has not yet been published as an RFC or equivalent stable document. Implementations of provisional types are encouraged but should expect specification changes during the provisional period.

Stable. The type’s specification has been published as an RFC or equivalent stable document. Implementations rely on a stable contract; specification changes after the Stable transition follow normal protocol-evolution procedures (errata, new versions, deprecation).

Deprecated. The type’s specification has been superseded or withdrawn. Deprecation does not suspend the membership criterion of Section 3.2; a deprecated type whose specification still satisfies the criterion remains a family member in good standing for backward-compatibility purposes, while a deprecated type whose specification no longer satisfies the criterion enters the contradiction state of Section 3.5. The Notes field SHOULD reference the superseding specification or document the reason for withdrawal. Verifiers MAY continue to support deprecated types for backward compatibility but SHOULD log their use distinctly to surface candidates for migration.

A type’s Status field is updated by the working group as the type’s specification matures, by the same Expert Review process specified in Section 8.2. A status transition MUST NOT change a registered type’s field-scope classifications (per Section 4.2) or its distinguishing field (per Section 5.4); changes to either are working-group revisions of the type’s specification, not administrative status updates.

8.5. Updates to This Document

Future revisions of this document that change the registry structure, the registration policy, or the Expert Review criteria are working-group revisions and MUST be processed by the working group through the same procedures that produced the current document. Editorial revisions that do not change the registry’s substantive structure are patch-level revisions per Section 6.5.

A future revision that adds a new scope category to Section 4.2.3 implicitly affects the Expert Review criteria of Section 8.3 (criterion 4 is updated to include the new category). Such a revision MUST update Section 8.3 explicitly so the criteria remain self-contained.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[VI-CONSTRAINTS] Verifiable Intent Working Group, "Verifiable Intent — Constraint Type Definitions and Validation Rules", Work in Progress, Internet-Draft, v0.1-draft, February 2026, <https://verifiableintent.dev/spec/constraints/>.

[VI-CREDENTIAL-FORMAT] Verifiable Intent Working Group, "Verifiable Intent — Credential Format Specification", Work in Progress, Internet-Draft, v0.1-draft, February 2026, <https://verifiableintent.dev/spec/credential-format/>.

9.2. Informative References

[ENVIRONMENT-MARKET-STATE] Headless Oracle Project, "Verifiable Intent — External State Attestation Constraint Proposal (environment.market_state)", Work in Progress v0.5.11-draft, 2026.

[ENVIRONMENT-WALLET-STATE] Borthwick, D., "Verifiable Intent — Wallet State Attestation Constraint Proposal (environment.wallet_state)", Work in Progress v0.6.5-draft, 2026.

[RFC-SD-JWT] Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)", Work in Progress, Internet-Draft, draft-ietf-oauth-selective-disclosure-jwt-22, 2025, <https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/rfc/rfc1918>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <https://www.rfc-editor.org/rfc/rfc6982>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7646] Ebersman, P., Kumari, W., Griffiths, C., Livingood, J., and R. Weber, "Definition and Use of DNSSEC Negative Trust Anchors", RFC 7646, DOI 10.17487/RFC7646, September 2015, <https://www.rfc-editor.org/rfc/rfc7646>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

[RFC8914] Kumari, W., Hunt, E., Arends, R., Hardaker, W., and D. Lawrence, "Extended DNS Errors", RFC 8914, DOI 10.17487/RFC8914, October 2020, <https://www.rfc-editor.org/rfc/rfc8914>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

Appendix A. Reference Type Specifications

This appendix records the constraint type specifications that have proposed registration in the environment.* namespace at the time of this document's publication. The appendix is informative, per [RFC6982].

This appendix is intended to be removed by the RFC Editor before final publication, per [RFC6982], Section 2. The version-control history of this document retains the appendix permanently as evidence of the family's running-code state during working-group review.

A.1. environment.market_state

[ENVIRONMENT-MARKET-STATE] proposes environment.market_state as a member of the environment.* family. The type addresses market-session state at a named exchange, with an attestation issued by an exchange-state oracle and verified against an [RFC8615] well-known key registry.

The reference implementation associated with [ENVIRONMENT-MARKET-STATE] is documented in that specification per [RFC6982] in its own Implementation Status appendix; this document does not duplicate that content.

Conformance to this document by [ENVIRONMENT-MARKET-STATE] at v0.5.11-draft covers Sections 3.1 through 3.4 (family definition and addition rules), Section 4.1 (family-wide fields), Section 4.2 (field-scope taxonomy), Section 4.3 (algorithm-agility framework), Section 5 (family composition, including conjunction, completeness, and per-member diagnostic output), Section 6 (register discipline), and the applicable subsections of Section 7 (Security Considerations).

A.2. environment.wallet_state

[ENVIRONMENT-WALLET-STATE] proposes environment.wallet_state as a member of the environment.* family. The type addresses on-chain wallet state under a specified condition predicate, with an attestation issued by an on-chain-state attestation issuer and verified against an [RFC7517] JWKS.

The reference implementation associated with [ENVIRONMENT-WALLET-STATE] is documented in that specification per [RFC6982] in its own Implementation Status appendix; this document does not duplicate that content.

Conformance to this document by [ENVIRONMENT-WALLET-STATE] at v0.6.5-draft covers Sections 3.1 through 3.4 (family definition and addition rules), Section 4.1 (family-wide fields), Section 4.2 (field-scope taxonomy), Section 4.3 (algorithm-agility framework), Section 5 (family composition, including conjunction, completeness, and per-member diagnostic output), Section 6 (register discipline), and the applicable subsections of Section 7 (Security Considerations). [ENVIRONMENT-WALLET-STATE] additionally specifies

cross-chain temporal consistency at its own Section 6.9; the family-wide implications of cross-chain temporal consistency are out of this document's scope and are addressed at the type-specification layer.

A.3. Disclaimer

Listing in this appendix does not imply endorsement of either type specification by the editors of this document, by the working group, or by IANA. Each type specification is independently reviewed under the registration process of Section 8. Verifiers MUST independently verify attestations from any implementation per the verification algorithm of the implementation's type specification.

Appendix B. Reserved

This appendix is reserved for a future revision of this document. Per Section 6.7 Q3, broader family-charter material — provider neutrality, peer-author coordination discipline, patch-level versioning conventions, lockstep commit patterns across sibling specifications, and a catalog of register-discipline instances across registered family members — may be consolidated under this appendix in a future revision, may be relocated to a separate appendix, or may be distributed across the relevant specification sections. The choice is a working-group decision the current revision does not foreclose.

This appendix is informative.

Acknowledgments

The authors thank the Verifiable Intent Working Group for the host specification on which this family-definition document depends. The authors additionally acknowledge the contributors and reviewers of [ENVIRONMENT-MARKET-STATE] and [ENVIRONMENT-WALLET-STATE], whose parallel review work converged the family-wide vocabulary, composition discipline, and register practices that this document formalises.

Authors' Addresses

Douglas Borthwick
InsumerAPI
New Canaan, CT
United States of America
Email: douglas@insumermodel.com
URI: https://insumermodel.com

Michael Msebenzi
Headless Oracle
Midrand
South Africa
Email: info@bytecraftresults.com
URI: https://headlessoracle.com