LARRI
VERIFICATION PROTOCOL

Ledger of Attested Records and Receipts for Integrity

Volume II of V

LARRI Artifact, Encoding, and Trust Contract Specification

Version 0.1.0

Purpose. Exact public contracts for interoperable LARRI artifacts, semantic-contract identity, strict decoding, canonical bytes, commitments, attestations, external trust inputs, prepared-directory bundles, resource limits, invocation refusal, and normalized verification results. These contracts define the machine-visible protocol boundary without prescribing internal software design or proprietary decision methods.

Property	Value
Document type	Normative artifact, encoding, and trust contract specification
Publication set	LARRI 0.1.0, Volume II of V
Protocol	LARRI - Ledger of Attested Records and Receipts for Integrity
Version	0.1.0
Status	Experimental
Use boundary	Synthetic laboratory and supervised evaluation only
Publication date	27 July 2026
Official website	https://larriverification.org

Assurance boundary. LARRI verifies the exact integrity, attribution, scope, and deterministic-verification properties defined by the selected target and profiles under explicitly supplied trust. It does not establish factual truth, human or organizational identity, authority, completeness, independently trusted time, clinical correctness, legal sufficiency, regulatory compliance, production readiness, universal absence of sensitive information, or correctness of the private decision process.

Normative specification

Contents

Document relationship. Volume I controls assurance meaning, roles and trust boundaries, verification targets, the eleven-stage dependency order, result meaning, and non-goals. This volume owns exact fields, schemas, lexical forms, canonical bytes, signature messages, trust objects, semantic-contract identity, prepared-directory bundle rules, resource limits, refusal representation, and normalized-result bytes. Volume III owns optional profile and extension semantics. Volume IV owns conformance evidence, vectors, reports, claims, and publication gates. Volume V is informative.

1. Authority, scope, semantic contracts, and invocation

This volume defines the public machine-visible contracts required to create and independently verify LARRI 0.1.0 artifacts. It is authoritative for exact field names, types, lexical forms, canonical byte sequences, identity calculations, signature inputs, trust-input forms, prepared-directory inventory rules, resource-limit records, invocation-refusal representation, check and refusal ordering, and normalized result bytes.

It does not define how an application reaches a decision, stores private evidence, structures software, selects a product workflow, or implements proprietary policies, models, prompts, thresholds, or decision mathematics.

1.1 Public source authority and closed objects

LARRI-010-ART-SCOPE-001 Every protocol-owned object MUST validate against exactly one registered top-level schema for the selected protocol and schema version.

LARRI-010-ART-SCOPE-002 Unknown members MUST be rejected unless the applicable registered schema expressly defines a closed nested object or a separately registered result schema. Base objects define no open member namespace.

LARRI-010-ART-SCOPE-003 Normative requirement identifiers MUST be globally unique across the LARRI 0.1.0 publication set and MUST use ASCII hyphens.

LARRI-010-ART-SCOPE-004 Every requirement reference in schemas, registries, profile records, vectors, reports, or gates MUST resolve to exactly one normative requirement definition.

Conflict rule. A publication set containing conflicting normative instructions for the same input and scope cannot support a conformance claim. Implementations do not resolve conflicts through aliases, document-order precedence, fallback parsing, reference-code behavior, or private tie-breakers.

1.2 Semantic-contract manifest and contracts_sha256

The semantic-contract layer is a prepared directory rooted at the fixed control file contracts.larri.json. It contains only the public semantic sources needed to determine verification behavior. The control file is canonical JSON and is not listed in its own files array. Invocation establishment applies the published baseline JSON limits, prepared-directory path protections, inclusive entry maximum, and inclusive total-byte maximum before the semantic contracts are accepted; operator-selected overrides do not alter this bootstrap safety boundary. For this semantic root, the entry count includes contracts.larri.json and every listed regular file, and the total-byte count includes all of those exact bytes.

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "schema_id": "larri/semantic-contract-manifest",
  "schema_version": "0.1.0",
  "protocol_id": "larri",
  "protocol_version": "0.1.0",
  "files": [
    {
      "path": "schemas/decision-receipt.schema.json",
      "contract_class": "schema",
      "byte_length": 1234,
      "sha256": "<64 lowercase hex>"
    }
  ]
}
```

Member	Type	Contract
schema_id	string	Constant larri/semantic-contract-manifest.
schema_version	string	Constant 0.1.0.
protocol_id	string	Constant larri.

Member	Type	Contract
protocol_version	string	Constant 0.1.0.
files	array	Non-empty, duplicate-free array sorted by normalized path. Each entry is a closed object.
files[].path	normalized_path	Path within the semantic-contract root; the control file is excluded.
files[].contract_class	ASCII enum	Exactly schema, registry, requirements, profile, extension, or gate.
files[].byte_length	integer	Exact non-negative file byte count.
files[].sha256	sha256_hex	SHA-256 over the exact listed file bytes.

contracts_sha256 is SHA-256 over the exact final canonical bytes of contracts.larri.json. The manifest does not contain contracts_sha256, vector_set_sha256, a release-envelope digest, or any self-reference. The selected contracts digest is supplied independently and compared with the recomputed value.

LARRI-010-ART-CONTRACTS-001 The semantic-contract manifest MUST list every semantic-contract file exactly once, MUST use closed file scope within its root, and MUST bind each file by exact length and SHA-256 digest.

LARRI-010-ART-CONTRACTS-002 The semantic-contract layer MUST contain schemas, registries, the normative requirements inventory, semantic profile and extension records, and non-release-specific gate definitions, and MUST exclude vectors, fixtures, frozen expected results, vector_set_sha256, conformance reports, report digests, publication metadata, and release-envelope identities.

LARRI-010-ART-CONTRACTS-003 A target-scoped verifier result MUST NOT exist until the semantic-contract manifest, its closed file set, every listed length and digest, and the selected contracts_sha256 have been established.

One-direction dependency. Semantic contracts may be referenced by conformance evidence. Semantic contracts MUST NOT contain evidence or release identities that create a reverse dependency. Release authentication is defined outside the semantic-contract identity.

1.3 Invocation establishment and invocation refusal

1 Establish the semantic-contract layer and contracts_sha256 under Section 1.2.

2 Compare the operator-selected target token as exact ASCII against the enabled verification-target registry.

3 After those two steps succeed, create the target-scoped verification context and the check set for that enabled target.

4 Evaluate the required verification-configuration object under the target-scoped Stage 1 through Stage 3 rules in Section 3.3 before processing artifact content under selected limits.

MACHINE-VALID EXAMPLE. This is the canonical logical object before canonical JSON serialization.

```
{
  "schema_id": "larri/invocation-refusal",
  "schema_version": "0.1.0",
  "protocol_id": "larri",
  "protocol_version": "0.1.0",
  "contracts_sha256": null,
  "code": "SEMANTIC_CONTRACTS_NOT_ESTABLISHED"
}
```

Member	Contract
schema_id	Constant larri/invocation-refusal.
schema_version	Constant 0.1.0.
protocol_id	Constant larri.
protocol_version	Constant 0.1.0.
contracts_sha256	Null only when semantic contracts were not established; otherwise the established digest.
code	Exactly one enabled invocation-code token.

Invocation code	contracts_sha256	Meaning
SEMANTIC_CONTRACTS_NOT_ESTABLISHED	null	The semantic-contract root, manifest, closed file set, or selected digest could not be established.
UNSUPPORTED_VERIFICATION_TARGET	sha256_hex	Semantic contracts were established, but the operator-selected target was not exactly one enabled target token.

LARRI-010-ART-INVOKE-001 An invocation refusal MUST use the closed larri/invocation-refusal object, MUST use exactly one enabled invocation code, and MUST NOT include the requested target text, artifact bytes, paths, host diagnostics, exceptions, or free-form messages.

LARRI-010-ART-INVOKE-002 An unsupported operator-selected target MUST be rejected before target-scoped verification, MUST NOT be mapped to another target, and MUST NOT produce a fabricated larri/verifier-result.

Once semantic contracts and an enabled target are established, a missing or invalid verification configuration is a target-scoped protocol defect. It is represented through the applicable input_safety, strict_decoding, schema_recognition, or contract_resolution refusal and never through larri/invocation-refusal.

2. Common data model and lexical forms

Protocol-owned JSON values are limited to null, Boolean, string, signed 64-bit integer, array, and object. Fractions and exponent notation are not part of the data model. Unless a field is expressly declared set-like, array order is significant.

Name	Full-string lexical form	Meaning
sha256_hex	Exactly 64 lowercase ASCII hexadecimal characters.	Digest or content-derived identifier.
public_key_base64	Exactly 44 RFC 4648 standard Base64 characters with one terminal = and zero unused pad bits.	Exactly 32 decoded public-key bytes.
signature_base64	Exactly 88 RFC 4648 standard Base64 characters with two terminal == and zero unused pad bits.	Exactly 64 decoded signature bytes.
rfc3339_time	Exactly YYYY-MM-DDTHH:MM:SSZ or the same with either a numeric +HH:MM or -HH:MM offset; year 0001 to 9999; valid proleptic-Gregorian month and day; hour 00 to 23; minute and second 00 to 59; offset hour 00 to 23 and minute 00 to 59; no fraction or leap second.	Issuer-asserted time unless a profile establishes a narrower claim.
normalized_path	Relative NFC path using forward slashes and the complete Section 8.4 grammar.	Prepared-directory location only.
namespace	Lowercase DNS-style name, 1 to 253 ASCII characters.	Application naming boundary with no base trust effect.
namespaced_id	namespace + / + 1 to 128 lowercase ASCII local characters from [a-z0-9._/-], with no empty segment.	Application or extension identifier.
key_id	1 to 128 ASCII characters: first [A-Za-z0-9], remainder [A-Za-z0-9._:-].	Opaque label within one trust or signed policy object.
local_token	1 to 128 lowercase ASCII characters: first [a-z], remainder [a-z0-9._-].	Assertion type or local application token.
media_type	Exactly lowercase ASCII type/subtype. Each component is 1 to 63 characters, begins and ends [a-z0-9], and uses only internal [a-z0-9!#$^_.+-]. No parameters or whitespace.	Descriptive only.
registry token	Exact ASCII value enabled in one named closed registry.	Meaning comes only from that registry.

Namespace labels are separated by a period. Each label is 1 to 63 characters, begins and ends with an ASCII lowercase letter or digit, and may contain internal ASCII hyphens. namespaced_id local segments are separated by / and cannot be empty, a single period, or two periods. rfc3339_time uses the proleptic Gregorian leap-year rule: divisible by 4, except years divisible by 100 unless also divisible by 400.

LARRI-010-ART-LEX-001 Digest strings MUST use exactly 64 lowercase ASCII hexadecimal characters and MUST be compared as exact ASCII.

LARRI-010-ART-LEX-002 A registry-bound value MUST be compared as one exact full-string registry token; case folding, punctuation substitution, aliases, prefix matching, suffix matching, and heuristic normalization MUST NOT change its meaning.

LARRI-010-ART-LEX-003 Time values in core artifacts MUST NOT be represented as independently trusted time solely because they occur in a signed object.

LARRI-010-ART-LEX-004 A set-like array MUST reject duplicates and MUST use the exact registry or lexical ordering declared by its schema before canonicalization.

LARRI-010-ART-LEX-005 Every schema lexical grammar MUST match the complete string and MUST pair fixed-length grammars with exact minimum and maximum lengths so that a trailing newline, whitespace, or additional token cannot be accepted.

3. Strict decoding, resource limits, and configuration

3.1 Strict JSON decoding

- Decode UTF-8 only. Reject UTF-8, UTF-16, and UTF-32 byte-order marks.
- Reject malformed UTF-8, UTF-8 encodings of surrogate code points, escaped lone surrogates, and JSON-forbidden control characters.
- Decode string escapes before member-name comparison. Reject duplicate decoded member names before normalization.
- Normalize decoded member names to NFC for collision detection and reject distinct names that become equal.
- Accept only the JSON number grammar for integers. Reject a decimal point, exponent, NaN, Infinity, and every value outside -9223372036854775808 through 9223372036854775807.
- Enforce the selected byte, depth, member-count, array-length, directory-entry, and total-byte limits before uncontrolled allocation or materialization.
- Do not deserialize executable objects, code, macros, plugins, or language-specific object graphs.

LARRI-010-ART-JSON-001 Invalid encoding, duplicate decoded member names, and post-NFC member-name collisions MUST be rejected before schema evaluation.

LARRI-010-ART-JSON-002 A target-scoped input defect MUST produce the applicable deterministic REFUSE or execution classification rather than uncontrolled execution, unsafe traversal, partial PASS, or a fabricated identity.

LARRI-010-ART-JSON-003 Every accepted protocol integer MUST be within the inclusive signed 64-bit range, and the decoder MUST reject fractions and exponent notation at every depth.

3.2 Baseline resource-limit record

Limit member	Inclusive maximum	Meaning
protocol_json_max_bytes	5,242,880	Maximum bytes in one protocol-owned JSON input.
json_max_depth	64	Maximum array/object container depth; a top-level container has depth 1.
json_max_object_members	200,000	Maximum total member occurrences across all objects in one JSON input.
json_max_array_length	200,000	Maximum elements in any one JSON array.
bundle_max_entries	10,000	Maximum discovered regular files plus the two fixed control objects.
bundle_max_total_bytes	524,288,000	Maximum sum of exact byte lengths of every regular file, including the fixed controls.

Each published maximum is inclusive. The immediately greater value produces INPUT_SAFETY_REFUSED with the corresponding detail code, unless the public input contract permits execution_limit before a target-scoped result can be emitted.

LARRI-010-ART-RESOURCE-001 A conforming baseline execution MUST enforce exactly the enabled larri/core-baseline-limits-v1 record and MUST NOT claim a lower local limit as the published baseline.

LARRI-010-ART-RESOURCE-002 An operator MAY select a valid stricter configuration. Every selected valid nonbaseline configuration MUST be canonically digested and bound through configuration_sha256 whether or not a threshold is reached or the outcome changes. A target-scoped refusal before any valid configuration is selected MUST use null configuration_sha256.

LARRI-010-ART-RESOURCE-003 Every enabled baseline limit MUST have a boundary case at the inclusive maximum and a deterministic refusal or execution classification at the immediately greater value.

LARRI-010-ART-RESOURCE-004 The baseline resource-limit record MUST contain exactly the six members listed in this section and MUST NOT define base archive expansion, compression-ratio, or decompression behavior.

3.3 Verification-configuration object

MACHINE-VALID EXAMPLE. This example is intended to satisfy the stated object contract exactly.

```
{
  "schema_id": "larri/verification-configuration",
  "schema_version": "0.1.0",
  "limits_id": "larri/core-baseline-limits-v1",
  "overrides": {}
}
```

The object is closed. overrides may contain only the six registered limit members. An override value is an integer that is at least 1, strictly lower than the published maximum, and valid for that limit; bundle_max_entries is at least 2. An empty overrides object is the exact baseline configuration. The configuration input is decoded under the published baseline bootstrap limits; an override never governs parsing of the object that selects it. Artifact processing that depends on selected limits does not proceed until configuration resolution passes.

Configuration condition	Owning check	Public result	configuration_sha256
Missing	contract_resolution	CONTRACT_RESOLUTION_REFUSED / CONFIGURATION_MISSING	null
Unsafe, incomplete, over baseline bootstrap limit, or strict-decoding failure	Earliest applicable Stage 1 or 2 check	Applicable target-scoped refusal and safe detail code	null
Closed-schema, member, type, or value-bound failure	schema_recognition	SCHEMA_RECOGNITION_REFUSED / CONFIGURATION_SCHEMA_INVALID	null
Schema-valid but nonbaseline overrides are not valid stricter public limits	contract_resolution	CONTRACT_RESOLUTION_REFUSED / CONFIGURATION_NOT_STRICTER	null
Exact baseline object	contract_resolution PASS	No refusal	null
Valid nonbaseline object	contract_resolution PASS	No refusal	SHA-256 of exact canonical configuration bytes

configuration_sha256 is null for the exact baseline configuration or when target-scoped verification REFUSES before any valid configuration is selected; it is the SHA-256 digest of the exact canonical configuration bytes for every selected valid nonbaseline configuration.

Once a valid nonbaseline configuration is selected, its digest remains present on every later PASS or REFUSE branch.

3.4 Receipt byte intake and received-byte identity

The receipt target accepts one finite byte-addressable receipt input. The Verifier computes SHA-256 over the exact complete byte stream as received before JSON decoding. This digest remains available on later decoding, schema, trust, identity, signature, commitment, or profile refusal branches.

artifact_sha256 is null only when the finite stream did not complete, the complete bytes were not supplied to the Verifier, or an applicable execution limit ended input before a truthful digest could be computed. A protocol JSON byte-limit refusal does not erase a digest already computed from a complete finite input.

4. Canonicalization

The mandatory canonicalization identifier is larri/canonical-json-int-v1. It maps one accepted logical JSON value to one exact UTF-8 byte sequence. The algorithm pins Unicode Character Database 16.0.0 and Unicode Normalization Form C.

1. Decode every JSON string escape to Unicode scalar values under the strict decoder.
2. Normalize every object member name and every string value to NFC using Unicode Character Database 16.0.0.
3. Reject an object if two distinct decoded member names become equal after NFC normalization.
4. Preserve array order exactly.
5. Sort normalized object member names lexicographically by Unicode scalar-value sequence: at the first differing scalar the lower value sorts first, and an exact prefix sorts before the longer sequence.
6. Emit null, true, and false in lowercase and emit no insignificant whitespace.
7. Emit integers in base 10 with a minus sign only when negative, no plus sign, and no leading zero except zero. The accepted lexical value -0 emits as 0.
8. Escape quotation mark as \" and reverse solidus as \\. Use \b, \t, \n, \f, and \r for U+0008, U+0009, U+000A, U+000C, and U+000D.
9. Emit every other U+0000 through U+001F scalar as a lowercase four-digit \u escape. Do not escape solidus.
10. Emit every other Unicode scalar value directly and encode the resulting character sequence as UTF-8 without a byte-order mark.

LARRI-010-ART-CANON-001 Every object identified as canonical MUST produce exactly the bytes defined by larri/canonical-json-int-v1 under Unicode Character Database 16.0.0.

LARRI-010-ART-CANON-002 Implementations MUST reproduce the public Unicode, escaping, integer-boundary, duplicate-name, normalization-collision, negative-zero, BOM, and invalid-encoding consequences without relying on reference implementation output to complete the algorithm.

LARRI-010-ART-CANON-003 Canonicalization MUST operate on the accepted logical JSON value and MUST NOT replace a separately required digest of exact received artifact bytes.

Canonical input bytes. A schema may additionally require that received bytes already equal their canonical serialization. The bundle manifest and semantic-contract manifest have this requirement. Receipt, external-trust, and verification-configuration inputs are identified or signed through their specified canonical logical values but need not be received in canonical layout unless their schema says otherwise.

5. Decision receipts and commitments

5.1 Top-level receipt object

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "schema_id": "larri/decision-receipt",
  "schema_version": "0.1.0",
  "signed": { "...": "closed signed content" },
  "attestations": [ { "...": "closed attestation" } ]
}
```

Member	Type	Contract
schema_id	string	Constant larri/decision-receipt.
schema_version	string	Constant 0.1.0.
signed	object	Closed signed-content object.
attestations	array	Array of individually well-formed closed attestation objects; required count, membership, and extra-attestation behavior are evaluated under the signed policy.

LARRI-010-ART-REC-001 A receipt MUST contain exactly the four members declared by its schema and MUST contain every and only the attestations permitted by its signed signature policy.

5.2 Signed-content object

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "object_type": "larri/attested-assertion",
  "protocol_id": "larri",
  "protocol_version": "0.1.0",
  "canonicalization_id": "larri/canonical-json-int-v1",
  "receipt_id": "<64 lowercase hex>",
  "created_at": "2026-07-27T12:00:00Z",
  "assertion": { "...": "application assertion" },
  "commitments": [],
  "limitations": ["NO_COMPLETENESS_GUARANTEE"],
  "signer": { "...": "signed signer declaration" }
}
```

Member	Contract
object_type	Constant larri/attested-assertion.
protocol_id	Constant larri.
protocol_version	Constant 0.1.0.
canonicalization_id	Constant larri/canonical-json-int-v1 for the base schema.
receipt_id	Content-derived identity from Section 5.4.
created_at	Required issuer-asserted rfc3339_time.
assertion	Closed application-assertion wrapper.
commitments	Array of closed commitment objects; may be empty unless the selected operation requires one.
limitations	Duplicate-free array sorted by limitation-registry rank then ASCII.
signer	Closed signed signer and signature-policy declaration.

LARRI-010-ART-REC-002 The signed-content object MUST contain the exact constants, members, types, ordering constraints, and conditional fields required by the registered 0.1.0 schema.

5.3 Signed signer declaration

Object	Members	Rules
signer	signature_policy_id, attestors	Closed object. The baseline policy identifier is larri/one-ed25519-v1.
attestor entry	key_id, suite_id	Closed object. Baseline suite is larri/ed25519-sha256-v1.

The structural schema validates the array and each closed entry. Duplicate handling, signed membership, baseline count, correspondence with outer attestations, and policy satisfaction are evaluated by signed_binding and receipt_attestations in their normative stages rather than being collapsed into a generic schema refusal.

LARRI-010-ART-SIGNER-001 Every outer attestation key_id and suite_id used to satisfy the policy MUST exactly match one signed attestor entry.

LARRI-010-ART-SIGNER-002 The signed attestors array MUST reject duplicate key_id and suite_id pairs. The baseline one-ed25519 policy requires exactly one signed attestor entry.

5.4 Receipt identity

1. Copy the parsed signed-content object.
2. Remove receipt_id from the copy.
3. Canonicalize the remaining object using canonicalization_id.
4. Compute SHA-256 over those canonical bytes.
5. Encode the digest as lowercase hexadecimal and compare it with receipt_id as exact ASCII.

LARRI-010-ART-RID-001 Receipt identity MUST be non-recursive and MUST be recomputed independently by the Verifier.

LARRI-010-ART-RID-002 Receipt identity MUST NOT include the outer attestations array or depend on a final bundle manifest that inventories the receipt.

5.5 Application assertion

Member	Contract
namespace	Required namespace.
type	Required local_token.
value	Optional public scalar or closed application-owned object. Presence of null counts as supplied.
value_digest	Optional sha256_hex commitment to a private result object.
media_type	Optional media_type; permitted only with value_digest.

LARRI-010-ART-ASSERT-001 Exactly one of value or value_digest MUST be present in an application assertion.

LARRI-010-ART-ASSERT-002 Base verification MUST evaluate the integrity and attribution of the assertion wrapper and MUST NOT infer substantive correctness from namespace, type, media_type, value, value_digest, or a passing signature.

5.6 Commitment object

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "commitment_id": "input-1",
  "role": "example.org/source-record",
  "digest_algorithm": "sha-256",
  "digest": "<64 lowercase hex>",
  "byte_length": 1234,
  "media_type": "application/octet-stream",
  "canonicalization_id": null
}
```

Member	Contract
commitment_id	Required key_id-form label unique within the signed content.
role	Required enabled commitment-role token or namespaced_id.
digest_algorithm	Required enabled digest algorithm; base value sha-256.

Member	Contract
digest	Required sha256_hex over exact or registered canonical bytes.
byte_length	Optional non-negative integer count of bytes actually hashed.
media_type	Optional descriptive media_type with no trust effect.
canonicalization_id	Optional enabled identifier or null; null means exact supplied bytes.

LARRI-010-ART-COMMIT-001 A commitment establishes correspondence to identified bytes and MUST NOT be represented as proof of correctness, completeness, legality, confidentiality, or availability.

LARRI-010-ART-COMMIT-002 commitment_id values MUST be unique within one signed-content object. Commitment-array order remains significant.

LARRI-010-ART-COMMIT-003 A Verifier MUST recompute a commitment when the selected target or profile requires it and the necessary bytes are supplied.

LARRI-010-ART-COMMIT-004 The commitments check MUST report UNVERIFIED when optional signed commitments are present but the selected operation does not supply the bytes or evidence required to recompute them.

5.7 Time and outer metadata

created_at is part of the signed content and has integrity under a valid receipt attestation. It remains issuer-asserted time. asserted_at is optional outer envelope metadata and is not authenticated by larri/ed25519-sha256-v1.

LARRI-010-ART-TIME-001 A Verifier MUST NOT describe created_at or asserted_at as independently trusted time without a selected time-evidence profile and verified external evidence.

LARRI-010-ART-TIME-002 Unauthenticated outer time metadata MUST NOT affect PASS, primary-refusal selection, trust identity, or any trusted-time claim.

6. Attestations and signature policies

6.1 Baseline attestation object

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "suite_id": "larri/ed25519-sha256-v1",
  "key_id": "issuer-key-1",
  "signed_content_sha256": "<64 lowercase hex>",
  "signature": "<standard Base64 of 64 bytes>",
  "asserted_at": "2026-07-27T12:00:00Z"
}
```

Member	Contract
suite_id	Constant larri/ed25519-sha256-v1 for the baseline schema.
key_id	Must match one signed attestor entry and one accepted external-trust entry.
signed_content_sha256	SHA-256 of the recomputed final canonical signed-content bytes.
signature	signature_base64 encoding of exactly 64 bytes.
asserted_at	Optional unauthenticated rfc3339_time; no PASS or trusted-time effect.

6.2 larri/ed25519-sha256-v1

The Verifier canonicalizes the final signed-content object, computes SHA-256, compares the recomputed digest with signed_content_sha256, and verifies Ed25519 over the 32 raw digest bytes. It never verifies a digest accepted solely from the artifact.

Acceptance rule	Baseline requirement
Public key	Exactly 32 raw bytes; canonical compressed Edwards25519 encoding; invalid, identity, and small-order points rejected.
Signature	Exactly 64 bytes: canonical R encoding followed by scalar S.
Scalar S	Canonical integer strictly less than the Edwards25519 group order L.
Point R	Canonical compressed point; invalid, identity, and small-order points rejected.
Challenge	h = SHA-512(R_encoded || A_encoded || M) reduced modulo L, where M is the raw 32-byte recomputed SHA-256 digest.
Equation	Accept only [S]B = R + [h]A under the strict equation. Cofactored or permissive alternatives are not equivalent.

For these rules, the field prime is p = 2^255 - 19; the subgroup order is L = 2^252 + 27742317777372353535851937790883648493; and B is the standard Ed25519 base point. Canonical point decoding requires the encoded y-coordinate to be less than p, successful curve decoding, and byte-for-byte equality with the canonical re-encoding. The identity is rejected, and a point is small-order for this contract when multiplying it by 8 yields the identity.

LARRI-010-ART-SIG-001 All protocol paths using larri/ed25519-sha256-v1 MUST apply the same strict key, signature, scalar, point, challenge, and equation semantics.

LARRI-010-ART-SIG-002 A mismatch between the recomputed signed-content digest and signed_content_sha256 MUST refuse before signature acceptance is claimed.

Construction note. larri/ed25519-sha256-v1 is Ed25519 applied to the raw 32-byte SHA-256 digest of the final canonical signed content. It is neither direct-message Ed25519 over the JSON bytes nor RFC 8032 Ed25519ph. Its security additionally depends on SHA-256 collision resistance. Any direct-message, alternative-prehash, post-quantum, or hybrid construction requires a distinct registered suite and policy identifier.

6.3 Baseline signature policy

The baseline policy identifier is larri/one-ed25519-v1. It requires exactly one signed attestor entry using larri/ed25519-sha256-v1 and exactly one matching outer attestation.

LARRI-010-ART-POLICY-001 An artifact MUST NOT define, embed, or alter the external trust material used to satisfy its own signature policy.

LARRI-010-ART-POLICY-002 The baseline one-ed25519 policy MUST refuse a missing attestation, an extra attestation, a key or suite mismatch, a signed-content digest mismatch, an invalid signature, or an unsupported required suite.

7. External trust contract

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "schema_id": "larri/external-trust",
  "schema_version": "0.1.0",
  "entries": [
    {
      "key_id": "issuer-key-1",
      "suite_id": "larri/ed25519-sha256-v1",
      "public_key_encoding": "base64-raw",
      "public_key": "<Base64 of 32 bytes>",
      "fingerprint_algorithm": "sha-256",
      "fingerprint": "<64 lowercase hex>",
      "permitted_targets": ["receipt", "bundle", "profiled-bundle"]
    }
  ]
}
```

Entry member	Contract
key_id	Unique key_id within the object. Entries are sorted by exact ASCII key_id.
suite_id	Enabled signature suite for the supplied key bytes.
public_key_encoding	Constant base64-raw for the baseline suite.
public_key	public_key_base64 encoding of exactly 32 decoded bytes.
fingerprint_algorithm	Constant sha-256 for the base object.
fingerprint	SHA-256 over the exact decoded public-key bytes.
permitted_targets	Duplicate-free target tokens sorted in verification-target registry order.

The top-level object is closed and contains exactly schema_id, schema_version, and entries. entries is non-empty. Supplied but unused entries remain in trust_input_sha256 but do not appear in trust_fingerprints. Stage 4 does not decode an otherwise unused entry as a cryptographic key merely because it is present.

LARRI-010-ART-TRUST-001 External-trust objects MUST be supplied through an operator-controlled path, stream, environment binding, configuration channel, or other source outside the artifact being verified.

LARRI-010-ART-TRUST-002 At the stage where a key is required, the Verifier MUST decode that key, apply the exact suite key checks, recompute SHA-256 over the raw public-key bytes, and refuse malformed key material or a fingerprint mismatch before treating a signature as trusted. Stage 4 applies this rule only to a direct receipt key or the manifest-control key; Stage 9 applies it to each safely processed contained receipt key.

LARRI-010-ART-TRUST-003 key_id values MUST be unique within one external-trust object. Ambiguous or duplicate labels MUST refuse rather than use first-match behavior.

LARRI-010-ART-TRUST-004 A trust entry MUST NOT authorize a target absent from permitted_targets.

7.1 trust_input_sha256 and used fingerprints

trust_input_sha256 is null when trust input is missing, incomplete, exceeds an applicable safety limit before completion, or cannot pass strict JSON decoding and canonicalization. Once the complete logical JSON value strictly decodes and canonicalizes, trust_input_sha256 is SHA-256 over its canonical bytes and remains present even if later trust-schema, ordering, duplicate-label, key, fingerprint, target-permission, or required-key checks refuse.

trust_input_sha256 identifies the supplied trust value; it is not proof that any entry was accepted. trust_fingerprints contains only the duplicate-free lowercase fingerprints of keys actually used to satisfy required attestations, sorted as exact ASCII. Merely supplied, rejected, or unused entries are omitted.

LARRI-010-ART-OUT-007 trust_input_sha256 and trust_fingerprints MUST follow the availability and actual-use rules in this section and MUST NOT represent merely supplied key material as trusted or used.

Trust meaning. A matching fingerprint establishes continuity with the supplied public-key bytes and the target permission encoded in the trust input. It does not establish human or organizational identity, authority, role, licensure, delegation, legal status, revocation status, or independently trusted time.

7.2 Exact trust phase boundary

For the receipt target, Stage 4 validates the complete external-trust object and resolves the key and suite required by the direct receipt. For bundle and profiled-bundle, Stage 4 validates the complete external-trust object and resolves only the key and suite required by manifest.larri.sig.json. Object closure, entry ordering, and duplicate key_id checks apply to the complete trust object, but cryptographic key decoding, fingerprint comparison, target permission, and actual-use treatment are performed only for entries required at the current stage.

The Verifier MUST NOT read an unverified bundle file to discover a contained receipt key requirement before manifest identity, manifest attestation, closed scope, artifact_type, length, and digest checks pass. After Stage 8 safely validates a contained receipt and its signed binding, Stage 9 resolves that receipt's key and suite against the already supplied trust object. A missing match uses RECEIPT_ATTESTATION_REFUSED with RECEIPT_TRUST_ENTRY_MISSING. A matching entry that cannot satisfy suite, key, fingerprint, or selected-target permission checks uses RECEIPT_ATTESTATION_REFUSED with RECEIPT_TRUST_ENTRY_UNUSABLE. The fingerprint enters trust_fingerprints only after the corresponding required signature verifies successfully.

8. Prepared-directory bundle contract

8.1 Base subject and fixed controls

The base bundle and profiled-bundle subjects are prepared directories. They use the fixed root control names manifest.larri.json and manifest.larri.sig.json. No generic byte-stream bundle carrier, archive serialization, or canonical directory byte stream is accepted by the base target. A byte-stream bundle carrier requires a separately registered carrier extension with an exact mapping, subject identity, safety limits, and vectors.

LARRI-010-ART-MAN-001 A base bundle Verifier MUST select manifest.larri.json and manifest.larri.sig.json explicitly and MUST NOT use filename fallback, path-prefix discovery, media-type inference, or heuristic manifest selection.

LARRI-010-ART-CARRIER-001 The base bundle and profiled-bundle targets MUST accept only the prepared-directory subject defined by this section. A byte-stream bundle input MUST refuse as an unsupported subject form unless a selected registered carrier extension defines it.

LARRI-010-ART-CARRIER-002 A byte-reproducible carrier or cross-carrier equivalence claim requires a separately registered carrier extension, exact public mapping, claim boundary, and vector set. It MUST NOT be inferred from base bundle PASS or from an implementation reproducibility claim.

8.2 Bundle manifest

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "schema_id": "larri/bundle-manifest",
  "schema_version": "0.1.0",
  "protocol_id": "larri",
  "protocol_version": "0.1.0",
  "manifest_id": "<64 lowercase hex>",
  "extra_file_policy": "closed",
  "files": [
    {
      "path": "records/r1.json",
      "artifact_type": "larri/decision-receipt",
      "content_role": "control_text",
      "byte_length": 1234,
      "sha256": "<64 lowercase hex>"
    }
  ]
}
```

Member	Contract
schema_id	Constant larri/bundle-manifest.
schema_version	Constant 0.1.0.
protocol_id	Constant larri.
protocol_version	Constant 0.1.0.
manifest_id	Non-recursive identity defined in Section 8.3.
extra_file_policy	Constant closed.
files	Array sorted by exact normalized path; may be empty when no non-control files are declared. Control objects are not listed.
files[].path	normalized_path; location only, with no classification effect.
files[].artifact_type	Required string or null. Exact semantics in Section 8.6.
files[].content_role	Exactly one enabled content-role token.
files[].byte_length	Exact non-negative file byte count.
files[].sha256	SHA-256 over the exact final file bytes.

LARRI-010-ART-MAN-002 The manifest MUST use closed file scope and every entry MUST use a compatible artifact_type and content_role. Missing, extra, duplicate, role-invalid, type-role-incompatible, length-mismatched, or digest-mismatched files MUST prevent bundle or profiled-bundle PASS.

LARRI-010-ART-MAN-003 A receipt MUST NOT be required to bind the final manifest that inventories that receipt.

LARRI-010-ART-MAN-004 The files array MUST reject duplicate normalized paths under manifest_identity before manifest_id recomputation and MUST be sorted by exact normalized_path scalar order before canonicalization. Observed prepared-directory path collisions remain input_safety defects.

8.3 Canonical form, manifest identity, and bundle artifact identity

1. Strictly decode and validate manifest.larri.json under the registered schema and limits.
2. Canonicalize the complete logical manifest and require the received manifest bytes to equal those canonical bytes exactly.
3. Copy the manifest object, remove manifest_id, canonicalize the copy, compute SHA-256, and compare the lowercase digest with manifest_id.
4. Canonicalize the final complete manifest including manifest_id and compute SHA-256 over those exact bytes. That digest is manifest_sha256 and the prepared-directory artifact_sha256.

Manifest-declared duplicate normalized paths are detected and refused by manifest_identity before manifest_id recomputation. By contrast, collisions among observed filesystem paths are detected by input_safety before file content is opened. artifact_sha256 is null when a prerequisite before or within manifest_identity does not PASS, including missing, incomplete, over-limit, malformed, schema-invalid, duplicate-path, unresolved, noncanonical,

or manifest_id-mismatched manifest input. Once manifest identity passes, artifact_sha256 remains the recomputed final manifest digest on later signature, scope, file, receipt, commitment, or profile refusal branches. File defects do not create a replacement directory digest.

8.4 Detached manifest attestation

SCHEMATIC - NOT MACHINE-VALID. Placeholders illustrate member positions and are not accepted values.

```
{
  "schema_id": "larri/bundle-manifest-signature",
  "schema_version": "0.1.0",
  "protocol_id": "larri",
  "protocol_version": "0.1.0",
  "manifest_id": "<64 lowercase hex>",
  "signature_policy_id": "larri/one-ed25519-v1",
  "suite_id": "larri/ed25519-sha256-v1",
  "key_id": "issuer-key-1",
  "manifest_sha256": "<64 lowercase hex>",
  "signature": "<standard Base64 of 64 bytes>"
}
```

The signature is verified over the 32 raw manifest_sha256 bytes using the strict Ed25519 acceptance semantics in Section 6. The object is closed and the policy and suite values are constants.

LARRI-010-ART-MANSIG-001 The Verifier MUST recompute and compare manifest_id and manifest_sha256 before accepting the detached manifest signature. Manifest canonical-form or identity refusal has precedence over signature refusal.

LARRI-010-ART-MANSIG-002 The manifest-signature key_id MUST resolve to an accepted external-trust entry permitted for bundle or profiled-bundle, as selected.

8.5 Prepared-directory path and subject safety

- Paths use forward slashes and are relative. Leading slash, drive prefix, colon, backslash, empty segment, . segment, .. segment, and parent traversal are prohibited.
- Every path is valid UTF-8 and NFC. NUL, bidi controls, C0 or C1 controls, and trailing dot or space in a segment are prohibited.
- For each segment, the ASCII case-insensitive base name before the first period MUST NOT be CON, PRN, AUX, NUL, COM1 through COM9, or LPT1 through LPT9.
- Duplicate exact paths and NFC-equivalent paths are prohibited. For case-fold comparison, apply Unicode 16.0.0 default case folding to the complete NFC path, normalize the folded result to NFC, and prohibit equal folded results.
- Symbolic links, hard links, reparse points, junctions, aliases, device nodes, sockets, named pipes, and other link-like or non-regular entries are prohibited. Directories are structural and are not manifest files.
- The Verifier inspects metadata and selected limits before opening regular file content. No archive extraction behavior is part of the base target.

LARRI-010-ART-PATH-001 A prohibited, ambiguous, colliding, link-like, non-regular, or out-of-scope prepared-directory entry MUST refuse without root escape, unsafe traversal, or uncontrolled materialization.

8.6 artifact_type and content_role

content_role	Preparation and verification
control_text	artifact_type must be non-null and either resolve to an enabled artifact type that permits control_text or satisfy application namespaced_id with no base semantic effect. Base protocol use is larri/decision-receipt. Bytes are UTF-8 without BOM and LF-only.
evidence	artifact_type may be null, application-namespaced, or registered for evidence. Exact bytes are preserved; base LARRI does not infer transformation from path or media type.
binary	artifact_type may be null, application-namespaced, or registered for binary. Exact bytes, length, and digest are verified; no parsing is implied.
extension	artifact_type must be a non-null namespaced identifier. Selected registered semantics execute only when required by the target or profile; otherwise only generic integrity applies.

A string artifact_type beginning with larri/ must resolve to an enabled protocol-owned artifact type. Any other string must satisfy namespaced_id. A null value means no artifact type is asserted. The path member identifies location only and has no artifact-classification effect.

LARRI-010-ART-MAN-005 Every manifest file entry MUST contain artifact_type. Contained decision receipts MUST be discovered only when artifact_type == "larri/decision-receipt" and MUST NOT be discovered from a path prefix, filename, extension, directory convention, media type, content sniffing, or heuristic parsing. The base bundle target verifies every such declared entry and does not infer a receipt when no such entry is declared.

When no decision-receipt artifact is declared, the aggregate receipt_identity, signed_binding, and receipt_attestations checks PASS over the empty declared set. That status does not claim that a receipt exists or that omitted material is complete.

LARRI-010-ART-MAN-006 An incompatible artifact_type and content_role combination MUST produce BUNDLE_SCOPE_REFUSED with ARTIFACT_TYPE_ROLE_MISMATCH. A registered artifact-type record MUST declare every content role it permits.

LARRI-010-ART-ROLE-001 A Verifier MUST NOT repair line endings, encoding, content-role, artifact-type, or exact-byte violations in received bundle files.

LARRI-010-ART-ROLE-002 An unknown or unselected extension file MAY remain subject to generic manifest path, length, and digest verification, but its bytes MUST NOT execute extension behavior, contribute semantic assurance, satisfy a profile, or count as conformance to that extension.

9. Verification checks, refusals, and dependency behavior

9.1 Exact eleven-stage order

Order	Stage	Required effect
1	Input and subject safety	Apply selected limits and reject unsafe subject forms before unsafe processing.
2	Strict decoding and schema recognition	Strictly decode required JSON inputs and identify exactly one applicable top-level schema.
3	Contract and feature resolution	Resolve configuration semantics, versions, artifact types, canonicalization, policies, every selected profile or exact combination, and every required extension.
4	External trust	Validate the supplied trust object and only the direct receipt or manifest-control trust requirements known without reading unverified bundle files.
5	Manifest canonical form and identity	For bundle targets, require canonical bytes and recompute manifest_id and final digest.
6	Detached manifest attestation	Verify the detached signature only after manifest identity and digest match.
7	Closed bundle scope and file integrity	Verify prepared-directory scope, paths, roles, artifact types, lengths, and digests.
8	Receipt identities and signed bindings	Strictly validate required receipts, recompute receipt IDs, and validate duplicated metadata.
9	Receipt attestations	Resolve contained-receipt trust entries after safe receipt processing and apply exact signer, suite, policy, and signature semantics.
10	Commitments and selected profiles	Recompute required commitments and evaluate only conditions of profiles and extensions already resolved successfully at Stage 3.
11	Normalized result	Select the deterministic primary refusal and emit canonical result bytes.

An inapplicable stage contributes no check and no refusal. A failed or unsafe prerequisite blocks dependent checks. Verification stage 2 strictly decodes and recognizes every present root receipt, fixed control object, supplied external-trust object, and verification configuration needed before feature resolution; absence is reported by the later owning check when that absence is itself the defined fault. Every unknown, disabled, deprecated, inapplicable, missing, unsupported-combination, ambiguous-combination, or required-extension profile-resolution defect is owned by contract_resolution at Stage 3. The Stage 10 profiles check evaluates only the public conditions of records already resolved successfully. Bundle-contained files are not parsed as receipts or extensions before manifest attestation, closed scope, artifact_type classification, length, and digest checks pass, and no contained receipt is read early to satisfy Stage 4 trust. A Verifier may perform internal work earlier only when it cannot change public status, refusal, identity availability, or primary-refusal precedence.

9.2 Base check registry

Rank	Check ID	Stage	Targets	Requiredness	Prerequisite	Allowed status
10	input_safety	1	receipt, bundle, profiled-bundle	Required	None	PASS, REFUSE
20	strict_decoding	2	receipt, bundle, profiled-bundle	Required	input_safety PASS	PASS, REFUSE, SKIP
30	schema_recognition	2	receipt, bundle, profiled-bundle	Required	strict_decoding PASS	PASS, REFUSE, SKIP
40	contract_resolution	3	receipt, bundle, profiled-bundle	Required	schema_recognition PASS	PASS, REFUSE, SKIP
50	external_trust	4	receipt, bundle, profiled-bundle	Required	contract_resolution PASS	PASS, REFUSE, SKIP
60	manifest_identity	5	bundle, profiled-bundle	Required	external_trust PASS	PASS, REFUSE, SKIP
70	manifest_attestation	6	bundle, profiled-bundle	Required	manifest_identity PASS	PASS, REFUSE, SKIP
80	bundle_scope	7	bundle, profiled-bundle	Required	manifest_attestation PASS	PASS, REFUSE, SKIP
90	file_integrity	7	bundle, profiled-bundle	Required	bundle_scope PASS	PASS, REFUSE, SKIP
100	receipt_identity	8	receipt, bundle, profiled-bundle	Required	external_trust PASS for receipt; file_integrity PASS for bundle targets	PASS, REFUSE, SKIP
110	signed_binding	8	receipt, bundle, profiled-bundle	Required	receipt_identity PASS	PASS, REFUSE, SKIP
120	receipt_attestations	9	receipt, bundle, profiled-bundle	Required	signed_binding PASS	PASS, REFUSE, SKIP
130	commitments	10	receipt, bundle, profiled-bundle	Contextual	receipt_attestations PASS	PASS, REFUSE, SKIP, NOT_PRESENT, UNVERIFIED
140	profiles	10	profiled-bundle	Required	Stage 3 profile resolution PASS; commitments not REFUSE or SKIP	PASS, REFUSE, SKIP

LARRI-010-ART-CHECK-001 The normalized result checks array MUST contain every and only the checks enabled for the selected target and profiles, in exact check-registry rank order. A check MUST NOT execute when its prerequisite did not PASS or when execution would be unsafe.

LARRI-010-ART-OUT-008 A blocked check MUST use SKIP with null refusal and the registered PREREQUISITE_FAILED or UNSAFE_TO_EXECUTE detail code. An inapplicable check is omitted rather than represented as SKIP.

9.3 Check statuses

Status	Meaning
PASS	The check executed and its requirement was satisfied.
REFUSE	The check executed and produced a blocking target-scoped refusal.
SKIP	The check did not execute because a prerequisite failed or execution would be unsafe.
NOT_PRESENT	An optional subject was not supplied and the selected operation did not require it.
UNVERIFIED	Optional signed material was present, but required bytes or evidence were unavailable for independent evaluation.

For the contextual commitments check: an empty commitments array with no selected requirement is NOT_PRESENT; all applicable supplied commitments independently recomputed and satisfied is PASS; optional signed commitments left unevaluated because their bytes are unavailable is UNVERIFIED with EVIDENCE_UNAVAILABLE; a required commitment or required bytes absent, or any evaluated length, digest, or canonicalization failure, is REFUSE. When PASS does not depend on the contextual check, NOT_PRESENT or UNVERIFIED does not become a profile or commitment assurance claim.

9.4 Refusal registry and detail selection

Rank	Refusal code	Owning check	Safe public description
10	INPUT_SAFETY_REFUSED	input_safety	The selected subject could not be processed safely under the selected limits.
20	STRICT_DECODING_REFUSED	strict_decoding	One required JSON input failed strict decoding.
30	SCHEMA_RECOGNITION_REFUSED	schema_recognition	One required input failed exact schema or lexical validation.
40	CONTRACT_RESOLUTION_REFUSED	contract_resolution	A required version, identifier, target feature, profile, or extension could not be resolved.
50	EXTERNAL_TRUST_REFUSED	external_trust	The external-trust object or a direct receipt or manifest-control trust requirement could not be accepted.
60	MANIFEST_IDENTITY_REFUSED	manifest_identity	The bundle manifest was not canonical, contained a duplicate normalized path, or its non-recursive identity did not match.
70	MANIFEST_ATTESTATION_REFUSED	manifest_attestation	The detached manifest attestation did not satisfy the registered policy.
80	BUNDLE_SCOPE_REFUSED	bundle_scope	The prepared directory did not match the signed closed manifest scope.
90	FILE_INTEGRITY_REFUSED	file_integrity	One or more in-scope files failed role, length, or digest verification.
100	RECEIPT_IDENTITY_REFUSED	receipt_identity	One or more required receipts failed strict receipt decoding or identity recomputation.
110	SIGNED_BINDING_REFUSED	signed_binding	Signed and duplicated signer or policy metadata did not match.
120	RECEIPT_ATTESTATION_REFUSED	receipt_attestations	One or more required receipt attestations did not satisfy the registered policy.
130	COMMITMENT_REFUSED	commitments	A required commitment could not be recomputed or did not match.

Each REFUSE check contains its owning refusal code. The primary refusal is the lowest-stage, then lowest-check-rank blocking refusal. The refusals array lists the primary code first; safely completed sibling refusals follow by refusal-registry rank and then exact ASCII code. The array is duplicate-free.

The base refusal table contains no Stage 10 profile-selection refusal. A profiles check may REFUSE only with a profile-specific refusal code already resolved and enabled at Stage 3. Selection, status, applicability, exact-combination, ambiguity, and required-extension resolution defects always use CONTRACT_RESOLUTION_REFUSED.

If one check observes more than one defect, detail_code is the first applicable enabled detail code in the published detail-code registry order. The Verifier does not execute additional parsing or downstream checks solely to obtain another detail.

LARRI-010-ART-CHECK-002 A normalized result MUST use only enabled check, refusal, status, and detail-code tokens and MUST select primary and additional refusals exactly as defined by the stage, check, refusal, and detail-code registries.

LARRI-010-ART-REFUSAL-001 A check record with status REFUSE MUST contain its registered owning refusal code; every other status MUST contain null refusal. detail_code MUST be null or one enabled safe detail code applicable to that check and status.

10. Normalized result and invocation-refusal objects

10.1 larri/verifier-result

SCHEMATIC - NOT MACHINE-VALID. The object shape is exact, but placeholder strings are not accepted values.

```
{
  "schema_id": "larri/verifier-result",
  "schema_version": "0.1.0",
  "protocol_id": "larri",
  "protocol_version": "0.1.0",
  "verification_target": "receipt",
  "profiles": [],
  "artifact_sha256": "<64 lowercase hex or null>",
  "contracts_sha256": "<64 lowercase hex>",
  "trust_input_sha256": "<64 lowercase hex or null>",
  "trust_fingerprints": [],
  "limits_id": "larri/core-baseline-limits-v1",
  "configuration_sha256": null,
  "outcome": "REFUSE",
  "primary_refusal": "EXTERNAL_TRUST_REFUSED",
  "refusals": ["EXTERNAL_TRUST_REFUSED"],
  "checks": [ { "...": "closed check record" } ],
  "limitations": ["NO_COMPLETENESS_GUARANTEE"]
}
```

Member	Rule
schema_id	Constant larri/verifier-result.
schema_version	Constant 0.1.0.
protocol_id	Constant larri.
protocol_version	Constant 0.1.0.
verification_target	Exactly one enabled target token.
profiles	Duplicate-free resolved selected profile identifiers sorted by profile-registry rank then ASCII; empty for receipt and bundle and when no profile resolved.
artifact_sha256	sha256_hex or null under Section 10.2.
contracts_sha256	Always the established semantic-contract digest; never null.
trust_input_sha256	sha256_hex or null under Section 10.2.
trust_fingerprints	Duplicate-free actual used fingerprints sorted as ASCII.
limits_id	Constant larri/core-baseline-limits-v1 for base 0.1.0.
configuration_sha256	Null for the exact baseline configuration or a target-scoped REFUSE before any valid configuration was selected; otherwise digest of the selected valid nonbaseline object.
outcome	PASS or REFUSE.
primary_refusal	Null on PASS; enabled refusal code on REFUSE.
refusals	Duplicate-free deterministic refusal array.
checks	Closed check records in exact registry order.
limitations	Duplicate-free limitations sorted by limitation rank then ASCII.

configuration_sha256 is null for the exact baseline configuration or when target-scoped verification REFUSES before any valid configuration is selected; it is the SHA-256 digest of the exact canonical configuration bytes for every selected valid nonbaseline configuration.

For every target that makes no verified completeness claim, limitations includes NO_COMPLETENESS_GUARANTEE. A selected passing profile may remove or replace that limitation only under the exact semantic contract defined by Volume III.

10.2 Digest and nullability branch table

Field	Non-null condition	Null condition	Retention rule
contracts_sha256	Every target-scoped result	Never	Target-scoped verification does not begin until semantic contracts are established.
artifact_sha256 - receipt	Complete finite receipt bytes were obtained	Only when bytes were incomplete or safe execution ended before a truthful full-stream digest	Retained after later decoding, schema, trust, identity, signature, commitment, or profile refusal.
artifact_sha256 - prepared directory	All preceding required stages through external_trust passed, and manifest bytes were complete, strict, schema-valid, canonical, identifiers resolved, and manifest_id matched	Missing, incomplete, over-limit, malformed, noncanonical, unresolved, or identity-mismatched manifest	Equals SHA-256 of final canonical manifest bytes; retained on later refusal.
trust_input_sha256	Complete trust JSON strictly decoded and canonicalized	Missing, incomplete, over-limit, or not canonicalizable	Retained through later trust schema, duplicate, key, fingerprint, permission, or use refusal.
configuration_sha256	Any valid nonempty overrides object selected	The exact baseline configuration, or a target-scoped REFUSE before any valid configuration was selected	Once a valid nonbaseline configuration is selected, its digest is retained whether or not a threshold is reached, the outcome changes, or a later check refuses.

configuration_sha256 is null for the exact baseline configuration or when target-scoped verification REFUSES before any valid configuration is selected; it is the SHA-256 digest of the exact canonical configuration bytes for every selected valid nonbaseline configuration.

LARRI-010-ART-OUT-001 The normalized result MUST be canonicalized with larri/canonical-json-int-v1 before digesting, storage as an expected result, or conformance comparison.

LARRI-010-ART-OUT-002 PASS requires null primary_refusal, an empty refusals array, and PASS for every required check. REFUSE requires an enabled primary_refusal equal to the first refusals entry.

LARRI-010-ART-OUT-003 Normalized results MUST exclude absolute paths, hostnames, process identifiers, local timestamps, execution duration, stack traces, exception text, stdout, stderr, temporary paths, raw private evidence, and nondeterministic ordering.

LARRI-010-ART-OUT-004 The exact baseline configuration and every target-scoped refusal emitted before any valid configuration was selected MUST use null configuration_sha256. Every selected valid nonbaseline configuration MUST use the SHA-256 digest of its canonical complete configuration object and retain that digest on every later result branch.

LARRI-010-ART-OUT-005 A target with no selected profile MUST emit an empty profiles array rather than omitting the member or inserting a base-profile alias. profiled-bundle with no enabled profile selected MUST use CONTRACT_RESOLUTION_REFUSED with PROFILE_SELECTION_REQUIRED rather than invent a profile or defer selection failure to Stage 10.

LARRI-010-ART-OUT-006 A normalized result MUST NOT contain a zero digest, empty-input digest, fixed placeholder, synthetic directory identity, or other fabricated identity. Null is permitted only on the exact branches defined in this section.

10.3 Check-record object

MACHINE-VALID EXAMPLE. This example is intended to satisfy the stated object contract exactly.

```
{
  "id": "external_trust",
  "status": "REFUSE",
  "refusal": "EXTERNAL_TRUST_REFUSED",
  "detail_code": "TRUST_INPUT_MISSING"
}
```

The check-record object is closed and contains exactly id, status, refusal, and detail_code.

10.4 Canonical invocation refusal

larri/invocation-refusal is a separate closed object and is not a larri/verifier-result. Volume IV maps it to the invocation_refuse execution class. It is canonicalized with larri/canonical-json-int-v1.

11. Registries, profiles, extensions, and result boundaries

Registry	Purpose
artifact_types	Protocol-owned and registered extension artifact identifiers; permitted content roles and top-level schema bindings.
canonicalization_algorithms	Canonical byte mappings, Unicode data version, parameters, and applicability.
digest_algorithms	Digest identifiers, output encodings, and exact input semantics.
signature_suites	Key, signature, message construction, and strict acceptance rules.
signature_policies	Required membership, counting, ordering, and failure behavior.
verification_targets	Enabled targets, subject forms, required checks, permitted profiles, and bounded claims.
check_identifiers	Check rank, stage, target applicability, prerequisites, requiredness, statuses, refusals, and details.
content_roles	Prepared-directory file treatment and artifact-type compatibility.
commitment_roles	Core roles and application namespace rules.
profiles	Profile identifiers, status, targets, rank, and semantic record digest.
extensions	Artifact, algorithm, carrier, evidence, trust, result, and check extensions.
profile_combinations	Closed permitted exact ordered profile sets.
resource_limits	The exact six-member baseline record.
limitations	Safe public limitation identifiers and result order.
refusal_registry	Refusal code, stage, rank, owning check, applicability, and safe description.
detail_codes	Safe detail token, owning check, status applicability, and rank.
invocation_codes	Closed pre-target invocation refusal codes.

LARRI-010-ART-REG-001 An unregistered, disabled, deprecated, or inapplicable identifier required by the selected target or profiles MUST produce the applicable registered target-scoped refusal. An unsupported operator-selected target is handled only by the invocation-refusal contract.

LARRI-010-ART-REG-002 An unknown profile or extension MUST NOT silently alter trust, signed bytes, required checks, PASS, REFUSE, primary refusal, artifact classification, or normalized output.

LARRI-010-ART-REG-003 Every value used in a schema-declared closed-registry field MUST resolve to exactly one applicable registry entry. Application-namespaced fields are validated by their declared grammar and need not appear in a LARRI registry unless selected for semantic execution.

LARRI-010-ART-REG-004 Generated prose tables, registry snapshots, schemas, and package views MUST derive from one canonical authored public source and MUST be checked for exact parity. Reference implementation code and undocumented validators are not normative authority.

11.1 Extension handling in base objects and bundles

Base protocol objects are closed and define no generic unknown-member container. A registered extension uses a separately registered artifact schema, carrier, result schema, trust-input schema, or other exact contract. In a prepared directory, content_role extension permits an unselected artifact to remain subject to generic path, length, and digest integrity.

LARRI-010-ART-EXT-001 Opaque preservation or generic byte verification of an unknown or unselected extension MUST NOT be represented as execution, verification, conformance, or assurance of that extension.

LARRI-010-ART-RESULTEXT-001 A profile requiring additional operator-controlled trust or policy input MUST NOT be enabled until it registers either a distinct closed profile-specific result schema or another closed public result contract that retains the exact added trust and policy identities. It MUST NOT add unknown members to larri/verifier-result.

11.2 Profiled-bundle boundary

The profiled-bundle target remains enabled. At Stage 3 it must resolve one enabled profile or one enabled exact profile combination and every required extension. Unknown, disabled, deprecated, inapplicable, missing, unsupported-combination, ambiguous-combination, and required-extension resolution failures are contract_resolution refusals. Only after resolution succeeds may Stage 10 evaluate required profile artifacts, evidence, checks, claims, limitations, and other profile conditions. Exact combination uniqueness, claims, limitations, and additional trust semantics are owned by Volume III. No enabled profile may create an implicit trust input or a hidden result dependency.

12. Security, privacy, examples, and source authority

12.1 Security and privacy boundaries

Risk	Contract control	Residual limitation
Artifact self-trust	External trust supplied out of band; exact fingerprint comparison; target permissions.	Does not establish who controls the key or whether the controller is authorized.
Serialization divergence	Strict JSON, closed schemas, UCD 16.0.0 canonicalization, and frozen expected results.	Only enabled encodings and algorithms are interoperable.
Signature ambiguity	Strict Ed25519 point, scalar, message, and equation rules.	The baseline construction also depends on SHA-256 collision resistance.
Prepared-directory exploitation	Metadata scan, path grammar, prohibited links, closed scope, and exact six-member limits.	Implementation defects remain possible and require conformance testing.
Private-method disclosure	Opaque assertions and digest commitments.	Low-entropy commitment preimages may be guessable.
Result overclaim	Target, profile, identities, checks, refusals, and limitations are explicit.	Relying processes must interpret the bounded result correctly.

LARRI-010-ART-PRIV-001 A protocol artifact or ordinary conformance claim MUST NOT require disclosure of proprietary decision logic, source code, private evidence, model weights, thresholds, prompts, policies, or internal state.

LARRI-010-ART-PRIV-002 Normalized results and public refusal descriptions MUST NOT expose raw private evidence, sensitive matched values, private keys, signing seeds, secrets, host paths, or free-form exception text, and MUST NOT claim universal absence of sensitive information.

LARRI-010-ART-PRIV-003 A plain public digest of a low-entropy sensitive value MUST NOT be represented as confidentiality protection, anonymization, or proof of absence.

12.2 Example authority

An example is normative only to the extent that a normative rule expressly incorporates it. Machine-valid examples are designated and must pass the published schemas and registries. Schematic examples may show placeholders but cannot serve as validation fixtures or normative authorities.

LARRI-010-ART-EXAMPLE-001 Every example containing a placeholder, ellipsis, omitted member, synthetic digest, synthetic Base64 value, or nonfinal signature MUST be labeled SCHEMATIC - NOT MACHINE-VALID. Every example designated machine-valid MUST validate under exactly one intended schema and every applicable registry rule.

LARRI-010-ART-VERSION-001 A Verifier MUST refuse an unsupported schema, protocol, canonicalization, signature suite, target feature, or semantic-contract version and MUST NOT apply fallback field aliases, punctuation substitution, or implementation-private compatibility rules.

A. Closed schema catalogue

Top-level schema	Exact top-level members	Applicability
larri/semantic-contract-manifest	schema_id, schema_version, protocol_id, protocol_version, files	Invocation establishment only.
larri/invocation-refusal	schema_id, schema_version, protocol_id, protocol_version, contracts_sha256, code	Pre-target bounded refusal.
larri/decision-receipt	schema_id, schema_version, signed, attestations	Receipt target or contained receipt.
larri/external-trust	schema_id, schema_version, entries	Operator-controlled external input.
larri/verification-configuration	schema_id, schema_version, limits_id, overrides	Selected limits configuration.
larri/bundle-manifest	schema_id, schema_version, protocol_id, protocol_version, manifest_id, extra_file_policy, files	Prepared-directory logical subject.
larri/bundle-manifest-signature	schema_id, schema_version, protocol_id, protocol_version, manifest_id, signature_policy_id, suite_id, key_id, manifest_sha256, signature	Detached manifest attestation.
larri/verifier-result	17 exact members defined in Section 10.1	Target-scoped result only.

Closed-object discipline. Every listed protocol-owned object uses additionalProperties false or the equivalent exact closed-member rule. Nested protocol-owned signer, attestor, assertion wrapper, commitment, trust entry, manifest entry, and check-record objects are also closed. Application-owned assertion value objects are application data and do not create new base protocol members.

B. Baseline registry snapshot

Registry class	Enabled identifier or record
Protocol	larri
Protocol version	0.1.0
Canonicalization	larri/canonical-json-int-v1; Unicode 16.0.0
Digest	sha-256
Signature suite	larri/ed25519-sha256-v1
Signature policy	larri/one-ed25519-v1
Verification targets	receipt, bundle, profiled-bundle
Resource limits	larri/core-baseline-limits-v1; exactly six members
Content roles	control_text, evidence, binary, extension
Artifact types	Eight top-level protocol bindings; only larri/decision-receipt permits a base contained-file role
Limitation	NO_COMPLETENESS_GUARANTEE
Invocation codes	SEMANTIC_CONTRACTS_NOT_ESTABLISHED, UNSUPPORTED_VERIFICATION_TARGET

B.1 artifact_types registry

Artifact type identifier	Top-level schema binding	Base applicability	Permitted bundle content roles
larri/semantic-contract-manifest	larri/semantic-contract-manifest	Semantic-contract root control	None
larri/invocation-refusal	larri/invocation-refusal	Pre-target public refusal output	None
larri/decision-receipt	larri/decision-receipt	Direct receipt or declared contained receipt	control_text
larri/external-trust	larri/external-trust	Operator-controlled input only	None
larri/verification-configuration	larri/verification-configuration	Operator-controlled input only	None
larri/bundle-manifest	larri/bundle-manifest	Fixed prepared-directory control only	None
larri/bundle-manifest-signature	larri/bundle-manifest-signature	Fixed prepared-directory control only	None
larri/verifier-result	larri/verifier-result	Target-scoped public result only	None

Only an artifact type expressly permitted as a manifest-contained file declares a bundle content role. In the base registry, larri/decision-receipt alone permits control_text. The other seven protocol-owned top-level types are root controls, operator inputs, or public outputs and are not valid manifest file artifact types.

B.2 Check and refusal registries

Rank	Check ID	Stage	Targets	Requiredness	Prerequisite	Allowed status
10	input_safety	1	receipt, bundle, profiled-bundle	Required	None	PASS, REFUSE
20	strict_decoding	2	receipt, bundle, profiled-bundle	Required	input_safety PASS	PASS, REFUSE, SKIP
30	schema_recognition	2	receipt, bundle, profiled-bundle	Required	strict_decoding PASS	PASS, REFUSE, SKIP
40	contract_resolution	3	receipt, bundle, profiled-bundle	Required	schema_recognition PASS	PASS, REFUSE, SKIP
50	external_trust	4	receipt, bundle, profiled-bundle	Required	contract_resolution PASS	PASS, REFUSE, SKIP
60	manifest_identity	5	bundle, profiled-bundle	Required	external_trust PASS	PASS, REFUSE, SKIP
70	manifest_attestation	6	bundle, profiled-bundle	Required	manifest_identity PASS	PASS, REFUSE, SKIP
80	bundle_scope	7	bundle, profiled-bundle	Required	manifest_attestation PASS	PASS, REFUSE, SKIP
90	file_integrity	7	bundle, profiled-bundle	Required	bundle_scope PASS	PASS, REFUSE, SKIP
100	receipt_identity	8	receipt, bundle, profiled-bundle	Required	external_trust PASS for receipt; file_integrity PASS for bundle targets	PASS, REFUSE, SKIP
110	signed_binding	8	receipt, bundle, profiled-bundle	Required	receipt_identity PASS	PASS, REFUSE, SKIP
120	receipt_attestations	9	receipt, bundle, profiled-bundle	Required	signed_binding PASS	PASS, REFUSE, SKIP
130	commitments	10	receipt, bundle, profiled-bundle	Contextual	receipt_attestations PASS	PASS, REFUSE, SKIP, NOT_PRESENT, UNVERIFIED
140	profiles	10	profiled-bundle	Required	Stage 3 profile resolution PASS; commitments not REFUSE or SKIP	PASS, REFUSE, SKIP

Rank	Refusal code	Owning check	Safe description
10	INPUT_SAFETY_REFUSED	input_safety	The selected subject could not be processed safely under the selected limits.
20	STRICT_DECODING_REFUSED	strict_decoding	One required JSON input failed strict decoding.
30	SCHEMA_RECOGNITION_REFUSED	schema_recognition	One required input failed exact schema or lexical validation.
40	CONTRACT_RESOLUTION_REFUSED	contract_resolution	A required version, identifier, target feature, profile, or extension could not be resolved.
50	EXTERNAL_TRUST_REFUSED	external_trust	The external-trust object or a direct receipt or manifest-control trust requirement could not be accepted.
60	MANIFEST_IDENTITY_REFUSED	manifest_identity	The bundle manifest was not canonical, contained a duplicate normalized path, or its non-recursive identity did not match.
70	MANIFEST_ATTESTATION_REFUSED	manifest_attestation	The detached manifest attestation did not satisfy the registered policy.
80	BUNDLE_SCOPE_REFUSED	bundle_scope	The prepared directory did not match the signed closed manifest scope.
90	FILE_INTEGRITY_REFUSED	file_integrity	One or more in-scope files failed role, length, or digest verification.
100	RECEIPT_IDENTITY_REFUSED	receipt_identity	One or more required receipts failed strict receipt decoding or identity recomputation.
110	SIGNED_BINDING_REFUSED	signed_binding	Signed and duplicated signer or policy metadata did not match.
120	RECEIPT_ATTESTATION_REFUSED	receipt_attestations	One or more required receipt attestations did not satisfy the registered policy.
130	COMMITMENT_REFUSED	commitments	A required commitment could not be recomputed or did not match.

B.3 Detail-code registry

Rank	Owning check	Status	Detail code	Safe meaning
10	input_safety	REFUSE	RECEIPT_BYTES_INCOMPLETE	The finite receipt byte stream did not complete.
20	input_safety	REFUSE	UNSUPPORTED_SUBJECT_FORM	The selected target received a subject form not enabled by its contract.
30	input_safety	REFUSE	PROTOCOL_JSON_MAX_BYTES	A protocol-owned JSON input exceeded the inclusive byte limit.
40	input_safety	REFUSE	JSON_MAX_DEPTH	A protocol JSON value exceeded the inclusive container-depth limit.
50	input_safety	REFUSE	JSON_MAX_OBJECT_MEMBERS	A protocol JSON input exceeded the inclusive total object-member limit.
60	input_safety	REFUSE	JSON_MAX_ARRAY_LENGTH	A protocol JSON array exceeded the inclusive element limit.
70	input_safety	REFUSE	BUNDLE_MAX_ENTRIES	The prepared directory exceeded the inclusive entry-count limit.
80	input_safety	REFUSE	BUNDLE_MAX_TOTAL_BYTES	The prepared directory exceeded the inclusive total-byte limit.
90	input_safety	REFUSE	UNSAFE_PATH	A discovered path was absolute, traversing, malformed, or otherwise prohibited.
100	input_safety	REFUSE	PATH_COLLISION	Two discovered paths collided under exact, NFC, or case-fold comparison.
110	input_safety	REFUSE	PROHIBITED_LINK	A link-like or multiply linked entry was discovered.
120	input_safety	REFUSE	NON_REGULAR_ENTRY	A non-directory, non-regular, non-control filesystem entry was discovered.
130	strict_decoding	REFUSE	BOM_PRESENT	A prohibited byte-order mark was present.
140	strict_decoding	REFUSE	INVALID_UTF8	The input was not valid UTF-8.
150	strict_decoding	REFUSE	SURROGATE_CODE_POINT	A surrogate code point or lone surrogate escape was present.
160	strict_decoding	REFUSE	FORBIDDEN_CONTROL	A control character prohibited by JSON was present.
170	strict_decoding	REFUSE	DUPLICATE_MEMBER	Two decoded object member names were equal before NFC.
180	strict_decoding	REFUSE	NFC_MEMBER_COLLISION	Distinct decoded member names became equal after NFC.
190	strict_decoding	REFUSE	FLOATING_POINT_NUMBER	A number used a fraction or exponent.
200	strict_decoding	REFUSE	INTEGER_OUT_OF_RANGE	An integer was outside signed 64-bit range.
210	strict_decoding	REFUSE	INVALID_JSON_SYNTAX	The input failed the strict JSON grammar.
220	schema_recognition	REFUSE	NO_MATCHING_SCHEMA	The object matched no intended top-level schema.
230	schema_recognition	REFUSE	MULTIPLE_MATCHING_SCHEMAS	The object matched more than one intended top-level schema.
240	schema_recognition	REFUSE	SCHEMA_VALIDATION_FAILED	A required member, type, constant, bound, or conditional rule failed.
250	schema_recognition	REFUSE	CONFIGURATION_SCHEMA_INVALID	The verification-configuration object failed its exact closed schema or value bounds.
260	schema_recognition	REFUSE	UNKNOWN_MEMBER	A closed protocol-owned object contained an unknown member.
270	schema_recognition	REFUSE	LEXICAL_FORM_INVALID	A field failed its exact full-string lexical form.
280	contract_resolution	REFUSE	UNSUPPORTED_PROTOCOL_VERSION	The protocol version was not enabled.
290	contract_resolution	REFUSE	UNSUPPORTED_SCHEMA_VERSION	The schema version was not enabled.
300	contract_resolution	REFUSE	UNSUPPORTED_CANONICALIZATION	A required canonicalization identifier was not enabled.
310	contract_resolution	REFUSE	UNSUPPORTED_SIGNATURE_SUITE	A required signature suite was not enabled.
320	contract_resolution	REFUSE	UNSUPPORTED_SIGNATURE_POLICY	A required signature policy was not enabled.
330	contract_resolution	REFUSE	UNKNOWN_REQUIRED_IDENTIFIER	A required closed-registry identifier did not resolve.
340	contract_resolution	REFUSE	DISABLED_OR_INAPPLICABLE_IDENTIFIER	A required identifier was disabled or inapplicable.
350	contract_resolution	REFUSE	CONFIGURATION_MISSING	No verification-configuration object was supplied after an enabled target was established.
360	contract_resolution	REFUSE	CONFIGURATION_NOT_STRICTER	A nonbaseline configuration did not select valid stricter public limit behavior.
370	contract_resolution	REFUSE	PROFILE_SELECTION_REQUIRED	The profiled-bundle target selected no enabled profile.

Rank	Owning check	Status	Detail code	Safe meaning
380	contract_resolution	REFUSE	PROFILE_UNSUPPORTED	A selected profile was unknown, disabled, deprecated, or inapplicable.
390	contract_resolution	REFUSE	PROFILE_COMBINATION_UNSUPPORTED	The exact ordered profile set had no enabled combination.
400	contract_resolution	REFUSE	AMBIGUOUS_PROFILE_COMBINATION	The exact ordered profile set resolved ambiguously.
410	contract_resolution	REFUSE	REQUIRED_EXTENSION_UNSUPPORTED	A required extension was unknown, disabled, deprecated, or inapplicable.
420	external_trust	REFUSE	TRUST_INPUT_MISSING	Required external trust was not supplied.
430	external_trust	REFUSE	DUPLICATE_KEY_ID	The trust object contained a duplicate key label.
440	external_trust	REFUSE	KEY_MATERIAL_INVALID	A required public key failed its suite encoding or point checks.
450	external_trust	REFUSE	FINGERPRINT_MISMATCH	The recomputed public-key fingerprint did not match.
460	external_trust	REFUSE	TARGET_NOT_PERMITTED	A required trust entry did not permit the selected target.
470	external_trust	REFUSE	REQUIRED_KEY_NOT_FOUND	A key required for the direct receipt target or manifest control was absent.
480	manifest_identity	REFUSE	MANIFEST_MISSING	The fixed bundle manifest was not supplied.
490	manifest_identity	REFUSE	DUPLICATE_PATH	The manifest files array contained a duplicate normalized path.
500	manifest_identity	REFUSE	MANIFEST_NOT_CANONICAL	The received manifest bytes were not the exact canonical bytes.
510	manifest_identity	REFUSE	MANIFEST_ID_MISMATCH	The recomputed non-recursive manifest identity did not match.
520	manifest_attestation	REFUSE	MANIFEST_ATTESTATION_MISSING	The fixed detached attestation object was missing.
530	manifest_attestation	REFUSE	MANIFEST_METADATA_MISMATCH	Duplicated manifest identity, key, policy, suite, or digest metadata did not match.
540	manifest_attestation	REFUSE	MANIFEST_DIGEST_MISMATCH	The detached object did not contain the recomputed final manifest digest.
550	manifest_attestation	REFUSE	MANIFEST_SIGNATURE_INVALID	The detached manifest signature failed strict verification.
560	bundle_scope	REFUSE	MISSING_FILE	A manifest entry had no corresponding regular file.
570	bundle_scope	REFUSE	EXTRA_FILE	An unlisted regular non-control file was present.
580	bundle_scope	REFUSE	ARTIFACT_TYPE_ROLE_MISMATCH	artifact_type and content_role were incompatible.
590	file_integrity	REFUSE	BYTE_LENGTH_MISMATCH	A file length differed from the manifest.
600	file_integrity	REFUSE	FILE_DIGEST_MISMATCH	A file SHA-256 digest differed from the manifest.
610	file_integrity	REFUSE	CONTROL_TEXT_ENCODING_INVALID	A control_text file was not UTF-8 without BOM and LF-only.
620	receipt_identity	REFUSE	CONTAINED_RECEIPT_INVALID	A contained decision receipt failed strict receipt decoding or schema validation.
630	receipt_identity	REFUSE	RECEIPT_ID_MISMATCH	A recomputed receipt identity did not match.
640	receipt_identity	REFUSE	DUPLICATE_RECEIPT_ID	Two required contained receipts declared the same receipt identity.
650	signed_binding	REFUSE	SIGNED_POLICY_MEMBERSHIP_INVALID	The signed attestor membership, duplicate handling, or policy count was invalid.
660	signed_binding	REFUSE	SIGNER_BINDING_MISMATCH	Outer key or suite metadata did not match a signed attestor entry.
670	signed_binding	REFUSE	SIGNED_METADATA_MISMATCH	A duplicated signed member did not match.
680	receipt_attestations	REFUSE	RECEIPT_TRUST_ENTRY_MISSING	A contained receipt referenced no matching external-trust key entry.
690	receipt_attestations	REFUSE	RECEIPT_TRUST_ENTRY_UNUSABLE	A contained receipt trust entry failed suite, key, fingerprint, or target-permission use checks.
700	receipt_attestations	REFUSE	ATTESTATION_MISSING	A required receipt attestation was missing.
710	receipt_attestations	REFUSE	ATTESTATION_EXTRA	The selected baseline policy received an extra attestation.
720	receipt_attestations	REFUSE	SIGNED_CONTENT_DIGEST_MISMATCH	The recomputed signed-content digest did not match the attestation.
730	receipt_attestations	REFUSE	SIGNATURE_INVALID	A required receipt signature failed strict verification.
740	commitments	REFUSE	REQUIRED_COMMITMENT_MISSING	A selected target or profile required a commitment that was absent.

Rank	Owning check	Status	Detail code	Safe meaning
750	commitments	REFUSE	COMMITMENT_BYTES_MISSING	Bytes required for a mandatory recomputation were unavailable.
760	commitments	REFUSE	COMMITMENT_LENGTH_MISMATCH	The recomputed committed byte length did not match.
770	commitments	REFUSE	COMMITMENT_DIGEST_MISMATCH	The recomputed commitment digest did not match.
780	commitments	REFUSE	COMMITMENT_CANONICALIZATION_UNSUPPORTED	The required commitment canonicalization was unavailable.
790	global	SKIP	PREREQUISITE_FAILED	The check was blocked by an earlier failed prerequisite.
800	global	SKIP	UNSAFE_TO_EXECUTE	The check was skipped because execution would be unsafe.
810	global	UNVERIFIED	EVIDENCE_UNAVAILABLE	Optional signed material was present but could not be independently evaluated.

Generated view. The released canonical registry files control exact status, applicability, rank, prerequisites, and machine tokens. This appendix must pass exact parity checks against those files and cannot be edited independently.

C. Example designation rules

Designation	Use	Publication treatment
MACHINE-VALID EXAMPLE	Demonstrates one exact accepted or refused object.	Must validate under exactly one intended schema and all applicable registry rules.
SCHEMATIC - NOT MACHINE-VALID	Shows member positions or relationships using placeholders or ellipses.	Must never be used as a fixture, expected result, oracle, registry source, or proof of conformance.

The baseline verification-configuration and invocation-refusal examples in this volume are designated machine-valid logical objects. Every other JSON block containing angle-bracket placeholders, ellipses, or omitted nested members is expressly schematic.

End of specification. End of LARRI Artifact, Encoding, and Trust Contract Specification 0.1.0.