ALX Protocol

An open protocol for composable digital artifacts

Protocol version: 1
Maintainer: XNDR Labs
Last reviewed: 2026-07-14

Table of Contents

Abstract

ALX Protocol is an open protocol for composable digital artifacts. A Block binds opaque JSON-serializable `content` and normalized parent Block identities into deterministic content-only and lineage-bound identities. Parent relationships create a structural Graph that conforming implementations can verify and traverse when the required Block data is available. Open protocol design means that public rules, schemas, and canonical test vectors support independent implementation. Block creation and verification require no hosted service, vendor, blockchain, storage network, or economic mechanism. Artificial intelligence and agent workflows provide leading applications because those systems create lineage-critical handoffs at high volume. The same Block model applies to human-authored, machine-produced, software, media, research, and institutional artifacts. Applications retain responsibility for semantic meaning, truth, quality, permissions, safety, and acceptance.

Executive Summary

ALX Protocol is an open protocol for composable digital artifacts.

The protocol defines a minimal Block primitive for representing digital artifacts with deterministic identity, declared lineage, and local verification. Equivalent protocol inputs produce equivalent Block identities across conforming implementations.

Core protocol capabilities include:

- Block primitive — A Block contains four required protocol fields: `blockHash`, `contentHash`, `parentHashes`, and `content`. Protocol version 1 also permits the optional `protocolVersion` field.
- Deterministic identity — `contentHash` identifies canonicalized content. `blockHash` identifies canonicalized content together with normalized parentHashes.
- Structural composition — `parentHashes` declares upstream Block relationships that form derivation Graphs.
- Local verification — Conforming implementations recompute Block identities, validate declared parent relationships, and traverse available Graph data through shared protocol rules.
- Protocol boundary — ALX Protocol verifies identity, integrity, lineage, and Graph structure. Semantic interpretation, policy, governance, trust, and operational use remain external to the protocol.
- Open protocol — Public specifications, schemas, and canonical test vectors support independent conforming implementations.
- Extensible architecture — Storage systems, attestations, timestamps, indexes, registries, governance frameworks, and economic mechanisms may operate above the Block layer without changing protocol validity.

Terminology and Core Concepts

The following terminology distinguishes the protocol primitives, application-facing artifacts, structural relationships, and application-level interpretation used throughout this paper.

Term	Definition
Artifact	Digital material represented through one or more ALX Blocks
Block	ALX Protocol primitive binding content and declared parent Block identities into deterministic identity
Block Identity	Deterministic identifier uniquely representing a Block
Content Identity	Deterministic identifier representing canonical Block content
Lineage	Declared parent relationships describing how a Block derives from earlier Blocks
Composition	Representation of a digital artifacts through interconnected Blocks
Graph	Directed structure formed by parent relationships among Blocks
Derivation Graph	Complete Graph representing the declared derivation history of an artifact
Context	Application-layer interpretation derived from artifact content and Graph relationships
Verification	Independent validation of Block identity, integrity, and declared lineage
Verification Context	Protocol inputs defining Graph validation mode and external parent boundaries
Machine-consumable	Property enabling deterministic processing, verification, and traversal by independent software

Open Protocol	Public rules, schemas, and test vectors supporting independent conforming implementations
Canonicalization	Deterministic normalization producing a single canonical representation
Parent Block	Block referenced as a declared dependency of another Block
Parent Hash	Deterministic identifier referencing a parent Block
Structural Verification	Verification of Block identity and declared relationships without interpreting content semantics
Protocol Boundary	Separation between protocol guarantees and external application responsibilities
Opaque Content	Content treated as uninterpreted data by the protocol
Conforming Implementation	Implementation satisfying the protocol specification and conformance requirements
Extension	Optional protocol-defined or application-defined data outside the required protocol surface

ALX Big Picture Model

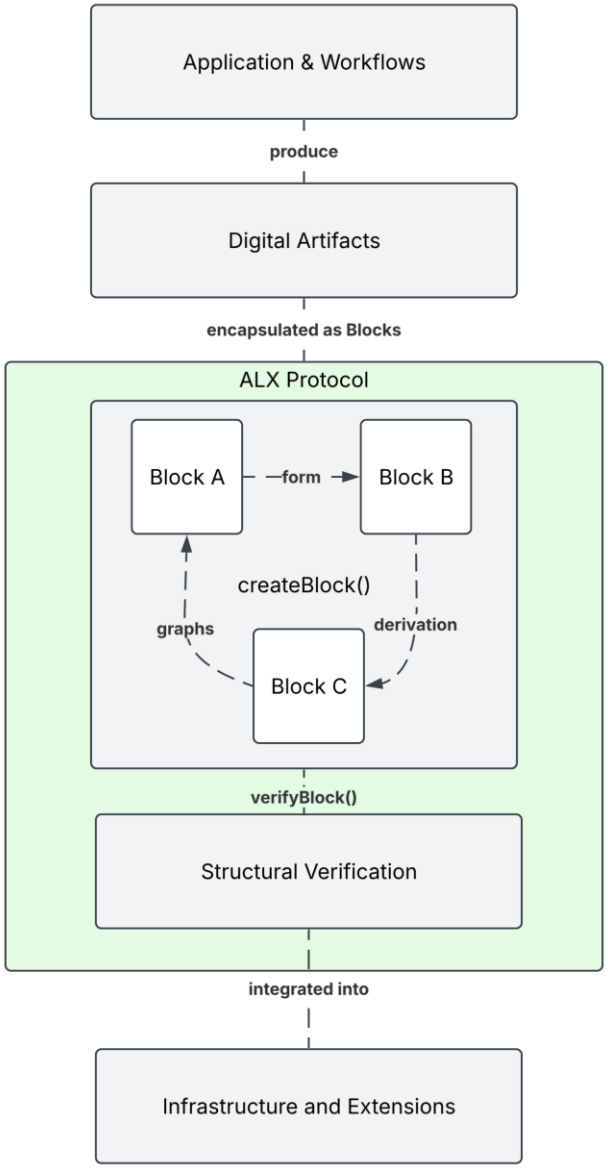

Figure 1. End-to-end ALX Protocol Lifecycle. Applications and workflows—including AI agents, research pipelines, software-development environments, enterprise systems, and APIs—produce digital artifacts such as source materials, code, datasets, model outputs, analyses, reviews, approvals, and decision records. In consequential workflows, conforming implementations represent these artifacts as deterministic, lineage-bound Blocks. Declared parent references compose Blocks into derivation Graphs that support structural verification of Block identity, content integrity, declared lineage, and Graph structure. External infrastructure and optional extensions can then ingest, store, index, publish, attest, anchor, discover, govern, and coordinate around Blocks, creating operational value through portability, reuse, auditability, attribution, interoperability, and decision support without changing core protocol identity or validity.

Part I—The Missing Layer in AI

Digital systems increasingly depend on artifacts produced across tools, models, agents, databases, workflows, and organizations. Artificial intelligence increases the volume and speed of this exchange, but the structural problem applies to human-authored, machine-produced, and AI-generated artifacts alike. A digital artifact is any information an application represents as opaque JSON-serializable content within a Block. Artifacts include source materials, model outputs, agent handoffs, workflow states, evaluations, decision records, media, software objects, and other information that downstream systems may reference, verify, or extend. Lineage is the declared structural relationship between a Block and its parent Blocks. Applications, AI systems, and implementers routinely need answers to questions such as:

- Which Block represents this artifact?
- Has the artifact `content` changed?
- Which upstream Blocks or artifacts are declared as dependencies?
- Which people, agents, models, systems, or workflows contributed to it creation?
- Which intermediate artifacts influenced the final result?
- Which derivation path connects this artifact to prior context?
- Can identity, integrity, and declared lineage be verified independently of the originating environment?

The artifact composition and lineage gap is the absence of a shared protocol for representing digital artifacts with deterministic identity, declared lineage, and independent verification. Without a common structural model, digital artifacts cannot be consistently composed, verified or reused across independent systems.

How Agentic Workflows Break Traceability

Multi-agent workflows produce a series of handoffs as artifacts move between models, agents, tools, and applications. A workflow may retrieve source material, summarize documents, extract claims, generate drafts, evaluate outputs, perform policy checks, and produce a final result. Each handoff transfers one or more artifacts to the next stage, creating a chain of structural dependencies that spans the entire workflow.

These handoffs carry more than content. They preserve the context that downstream systems rely upon, including source selections, intermediate analyses, evaluation results, review decisions, and other artifacts that influence subsequent work. As workflows become more distributed, handoffs increasingly cross application, organizational, and platform boundaries, making it difficult to preserve a consistent record of how the final artifact was produced.

Existing systems typically record workflow state, execution logs, or provenance within local environments. While valuable for operational purposes, these records rarely provide a portable representation of artifact identity, declared lineage, and derivation that independent systems can verify outside the originating environment.

Portable Lineage Enables Verifiable Sources

Retrieval-augmented generation (RAG) systems, copilots, and workflow agents increasingly depend on approved source material to produce reliable outputs. The quality of an artifact often depends not only on its content, but also on its declared relationships to the sources, policies, datasets, prompts, and prior decisions that informed its creation.

Portable lineage represents these relationships at the Block layer through explicit parent references. Rather than embedding provenance within application-specific workflows, a Block declares the upstream artifacts on which it depends using a portable, protocol-defined structure. This allows source relationships to move with the artifact as it passes between tools, vendors, organizations, and execution environments.

Downstream systems can independently verify Block identity, content integrity, and declared parent lineage through local computation. Because lineage is represented as protocol data rather than platform-specific metadata, source-grounded artifacts remain verifiable even after leaving the system that produced them.

Operational Context vs. Cross-Platform Trust

Existing systems provide valuable operational visibility. Logs capture runtime behavior. Dashboards present application state. Vendor APIs expose provider-specific records. Observability systems collect traces, while governance systems record review and approval events. These capabilities help organizations understand what occurred within a particular environment.

Operational visibility, however, does not necessarily produce portable trust. Most operational records remain tied to the systems that generated them, making it difficult for independent tools to verify artifact identity, lineage, and derivation after artifacts move across application, organizational, or platform boundaries.

ALX Protocol complements these systems by providing an artifact-level structure that travels with the artifact itself. Applications may represent context-bearing records as Blocks, allowing conforming implementations to verify Block identity and declared parent lineage independently of the originating environment.

By associating provenance with the artifact rather than the platform, ALX enables downstream systems to reference, verify, reuse, and extend digital artifacts after export, transfer, ingestion, summarization, or other forms of reuse.

Part II—Introducing the Block Model

ALX Protocol represents composable digital artifacts through a minimal Block model. Applications define the meaning of digital artifacts, while the protocol defines only the structural objects and relationships required for deterministic identity, lineage, and verification.

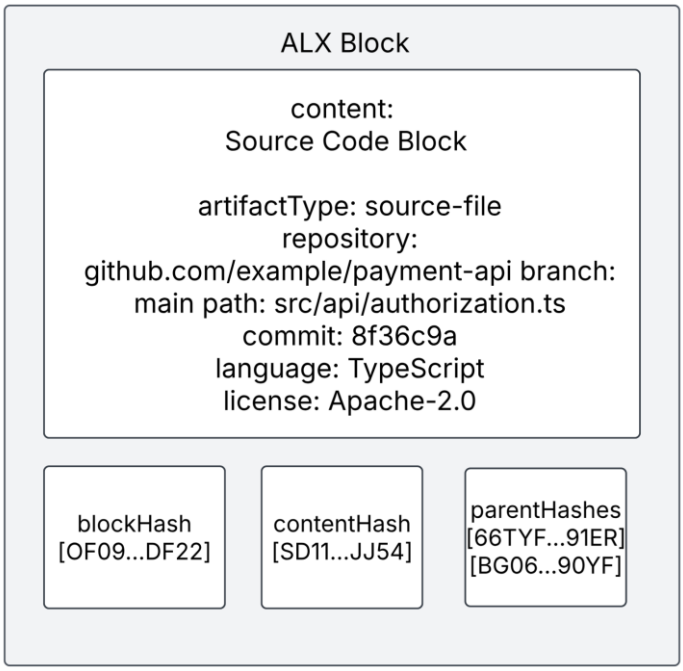

Figure 2. ALX Block Structure. Example ALX Block representing a source-code artifact. The Block records repository and commit metadata, while contentHash, parentHashes, and blockHash establish deterministic content identity and declared lineage.

The ALX Artifact Model

ALX Protocol provides a shared Block model for representing digital artifacts across application and infrastructure boundaries. Rather than standardizing application behavior or artifact semantics, the protocol defines a common structural representation that enables deterministic identity, declared lineage, and independent verification.

Each Block provides deterministic identity, explicit parent relationships, and the information required to verify its structural integrity. Together, interconnected Blocks represent the composition and declared derivation of digital artifacts without requiring applications to adopt a common interpretation of their content.

Because the model standardizes structure rather than semantics, AI-generated, human-authored, and machine-produced artifacts can participate in a common Graph across storage systems, execution environments, applications, verification tools, and organizations. External systems retain responsibility for meaning, governance, permissions, and trust, while ALX Protocol supplies the structural foundation that allows artifacts to remain portable and verifiable across system boundaries.

The Block Primitive

The Block is the fundamental primitive of ALX Protocol. It binds opaque artifact content and declared parent relationships into deterministic protocol identity.

A Block carries artifact content as opaque JSON-serializable data. The protocol does not inspect, interpret, classify, or validate the semantic meaning of that content. This opacity allows the same primitive to represent source materials, model outputs, agent handoffs, workflow states, evaluations, software objects, media, and other application-defined artifacts.

A Block also declares one or more parent Blocks that describe its structural dependencies. These parent relationships record how a Block derives from earlier Blocks without assigning semantic meaning to those relationships. Applications remain responsible for interpreting whether a parent represents a source, review, transformation, approval, or any other domain-specific relationship.

The Block distinguishes two forms of deterministic identity. One identity represents the artifact content alone, allowing independent verification of content integrity. The other represents both the content and its declared parent relationships, allowing independent verification of structural lineage. Together, these identities distinguish whether an artifact has changed, whether its lineage has changed, or both.

The Block is minimal by design. Storage, timestamps, attestations, signatures, schemas, indexes, registries, governance, and hosted services operate above the Block layer without affecting protocol identity or verification.

Inside a Block

A Block contains four required protocol fields that together provide deterministic identity, declared lineage, and independent verification.’

Field	Role	Identity Function
content	JSON-serializable artifact payload that is canonicalized before hashing	Used to derive both contentHash and blockHash
parentHashes	Normalized, deduplicated and lexicographically ordered list of declared parent blockHash values	Included in the blockHash identity input; excluded from contentHash
contentHash	Deterministic hash of canonicalized content	Identities and verifies the artifact payload independently of lineage

blockHash	Deterministic hash of canonicalized {content, parentHashes}	Identifies the Block and binds its content to its declared lineage

Table 1. Core ALX Block fields and their identity functions. The `contentHash` identifies canonical artifact content independently of lineage, while the `blockHash` binds that content to the normalized set of declared parent Block identities. This separation allows identical content to retain the same `contentHash` across different derivation contexts while receiving a distinct `blockHash` for each declared lineage.

How Blocks Compose Artifacts

Digital artifacts are represented through one or more interconnected Blocks. A simple artifact may consist of a single Block, while more complex artifacts may comprise multiple Blocks representing source materials, intermediate work, reviews, transformations, and final outputs. Together, these Blocks provide a structural representation of how an artifact was created and how it evolved over time.

Composition occurs when a Block declares one or more parent Blocks as structural dependencies. Each parent relationship records that the child Block derives from the declared parents without assigning semantic meaning to those relationships. Whether a parent represents a source, review, transformation, approval, or other dependency is determined by the application rather than the protocol.

These parent relationships form a directed Graph that represents the artifact's declared derivation history. Conforming implementations can verify available Blocks and traverse parent relationships using protocol data alone, while applications remain responsible for interpreting the meaning and validity of those relationships.

Summary

Artifacts are the application-facing assets represented through interconnected ALX Blocks. Declared parent relationships compose Blocks into verifiable Graphs that provide deterministic identity, lineage, and structural verification across independent systems while leaving application semantics outside the protocol.

Part III—Identity and Lineage Infrastructure

ALX Protocol represents digital artifacts through interconnected Blocks. Together, deterministic identity and declared parent relationships provide a portable structural model that independent systems can verify without interpreting application semantics.

Establishing Deterministic Artifact Identity

ALX Protocol establishes deterministic identity by hashing canonical protocol data. Because every conforming implementation canonicalizes protocol data in the same way, equivalent inputs always produce equivalent identities.

A Block has two complimentary deterministic identities:

1. `blockHash` = keccak256(canonicalize(content))
2. `blockHash` = keccak256(canonicalize({content, `parentHashes`}))

These identities serve different purposes:

- `contentHash` identifies the artifact content alone. Any change to the content produces a different `blockHash.`
- `blockHash` identifies the artifact content together with its declared parent lineage. Any change to the content or declared parent relationships produces a different `blockHash.`

This distinction allows conforming implementations to verify content integrity independently of structural lineage. Two Blocks may contain identical content and therefore share the same `contentHash`, while different declared parent relationships produce different `blockHash` values.

To ensure deterministic identity across independent implementations, the protocol normalizes `parentHashes` before computing `blockHash`. Normalization lowercases valid hashes, removes invalid entries, eliminates duplicates, and sorts the remaining values lexicographically. As a result, equivalent parent relationships always produce equivalent `blockHash` values.

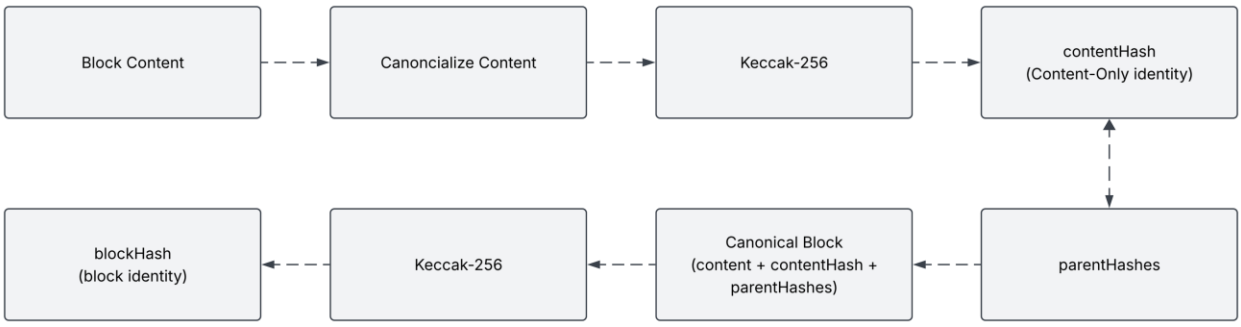

Figure 3. Deterministic identity generation in ALX Protocol. ALX derives two complementary identifiers from Block data. contentHash is computed from canonicalized content and identifies the artifact payload independently of lineage. blockHash is computed from the canonicalized combination of content and normalized parentHashes, binding the artifact payload and its declared lineage into a single deterministic Block identity. Consequently, Blocks containing identical content can share a contentHash while having different blockHash values when their declared parent relationships differ.

Verifying Individual Blocks

Block verification is a local computation performed entirely over Block data. A conforming implementation verifies a Block by canonicalizing its protocol fields, recomputing its deterministic identities, and comparing the recomputed values with the identities declared in the Block.

Block verification includes four core operations:

- Canonicalize block `content`
- Recompute contentHash as keccak256(canonicalize(content))
- Recompute blockHash as keccak256(canonicalize({ content, parentHashes }))
- Compare recomputed values with the declared `contentHash` and `blockHash`

A Block is structurally valid when its recomputed identities match the declared identities and the Block conforms to the required protocol structure.
Because verification depends only on protocol data and the canonicalization and hashing rules defined by ALX, it requires no network access, storage retrieval, blockchain confirmation, registry lookup, hosted service, or external authority.
Block verification establishes deterministic identity, content integrity, and declared lineage. It does not evaluate semantic truth, content quality, authorship, permissions, governance, compliance, or real-world identity, all of which remain the responsibility of applications and external systems.

Verifying Parent Lineage

Parent lineage verification confirms that a Block is structurally bound to its declared parent Blocks. Rather than evaluating the meaning of those relationships, verification establishes that the Block commits to a specific set of upstream Block identities.
A Block declares its upstream dependencies through `parentHashes`. Root Blocks contain an empty parent list, while derived Blocks declare one or more parent Block identities. During verification, a conforming implementation normalizes the declared parent set, recomputes `blockHash`, and confirms that the recomputed identity matches the declared value.
Because `blockHash` incorporates both the artifact content and its declared parent relationships, any change to either produces a different Block identity. This cryptographically binds declared lineage to the Block without assigning semantic meaning to the relationships themselves.
When parent Blocks are available, each can be verified independently before traversing the declared relationships to reconstruct the surrounding Graph. Applications remain responsible for determining whether those relationships represent valid sources, approvals, transformations, or other domain-specific dependencies.

Verifying the Complete Derivation Graph

A derivation Graph is formed when interconnected Blocks declare parent relationships. Each Block represents a node identified by its `blockHash`, while each parent reference defines a directed edge connecting a derived Block to one or more upstream Blocks. Together, these relationships represent the declared structural history of a digital artifact.

Derivation Graph verification extends individual Block verification across the connected Graph. A conforming implementation independently verifies each available Block, confirms that each Block is structurally bound to its declared parents, and validates that the declared relationships remain consistent throughout the Graph.

Once verified, the Graph can be traversed using the declared parent relationships. Starting from a descendant Block, a verifier can move upstream to inspect source materials, intermediate artifacts, reviews, policy decisions, agent handoffs, and other declared dependencies. When child relationships are indexed or otherwise available, traversal can also proceed downstream to analyze how earlier artifacts influenced subsequent work.

Derivation Graph verification establishes the structural integrity of the Graph. It confirms which Blocks are declared as dependencies, how those Blocks are connected, and whether each available Block maintains a valid deterministic identity. It does not evaluate semantic meaning, authorship, truth, policy compliance, or the correctness of the underlying workflow.

By representing lineage as protocol data rather than application metadata, ALX enables artifact histories to be independently verified, traversed, and reused across systems without requiring access to the environment in which they were created.

Maintaining Stable Identity Through Change

ALX Protocol maintains stable identity by treating every change as the creation of a new Block rather than the modification of an existing one. Because `blockHash` is derived from canonical content and declared parent relationships, any change to either produces a different deterministic Block identity.

Earlier Block identities remain valid and independently referenceable whenever the corresponding Block data is available. Rather than overwriting history, successive Blocks represent successive states of a digital artifact through explicit lineage.

This identity model distinguishes structural equivalence without relying on storage controls, append-only databases, access control, versioning systems, or hosted state management. Equivalent protocol data always produces the same identity, while different protocol data always produces a different identity.

Storage systems, applications, registries, indexes, and hosted services may delete, reorganize, archive, or relocate Block data according to their own requirements. These operations affect availability but do not change the deterministic identities defined by the protocol.

By separating identity from storage, ALX supports reproducibility, long-term referencing, and independent verification of reports, analyses, software artifacts, evaluations, agent handoffs, and other digital artifacts across changing systems.

Change	contentHash	blockHash	Reason
Canonical content unchanged; normalized parents unchanged	Same	Same	The Block identity input is identical
Canonical content unchanged; parents changed	Same	Different	The content is identical, but the declared lineage changed
Canonical content changed	Different	Different	The artifact content and Block Identity input changed
Parent order changed only	Same	Same	parentHashes are normalized into canonicalized order

Table 2. ALX identity invariants under content and lineage changes. `contentHash` remains stable when canonical content is unchanged, whereas `blockHash` binds canonical content to the normalized `parentHashes` set. Consequently, a change in declared lineage changes `blockHash` without changing `contentHash`. Reordering the same parent hashes does not affect Block identity because `parentHashes` are normalized into canonical order.

Separating Protocol and External Trust

ALX Protocol defines a minimal protocol layer responsible for deterministic Block identity and structural verification. The protocol layer consists of four required fields: content, `parentHashes`, `contentHash`, and `blockHash`.

Only these protocol fields participate in identity computation and protocol validity. `contentHash` is derived from canonicalized content, while `blockHash` is derived from canonicalized content together with the declared parent relationships. Every conforming implementation computes these identities using the same canonicalization and hashing rules.

Everything outside the protocol layer is external to Block identity. Storage systems, schemas, attestations, signatures, timestamps, indexes, registries, policy logic, hosted services, application metadata, user interfaces, and governance workflows may reference, store, interpret, or extend Blocks without affecting protocol validity.

This separation preserves a stable verification surface. Conforming implementations can independently verify Block identity, content integrity, and declared lineage using only protocol data, while applications and external systems remain free to provide domain-specific interpretation, trust signals, access control, discovery, compliance, and operational behavior.

The protocol defines structure. External systems define meaning and trust. By maintaining this boundary, ALX provides portable verification without constraining how applications store, govern, or interpret digital artifacts.

Defining the Verification Boundary

ALX Protocol verifies structure at the Block layer. Structural verification includes Block identity, content integrity, declared parent lineage, derivation Graph relationships, and structural path counts when the required Graph data is available

Verification confirms that `contentHash` matches the canonicalized content, `blockHash` matches the canonicalized content together with the declared parent relationships, and every declared parent reference is structurally bound into Block identity. When parent Blocks are available, conforming implementation can traverse the resulting derivation Graph to inspect declared dependency relationships.

Structural verification does not extend to semantic interpretation. Content quality, factual accuracy, model correctness, authorship, signer identity, storage availability, legal compliance, institutional trust, policy status, and real-world validity remain the responsibility of applications and external systems such as attestations, registries, governance frameworks, and policy engines.

This verification boundary allows ALX Protocol to remain domain-agnostic. The protocol provides deterministic identity, integrity, lineage, and structural verification, while applications and external systems provide interpretation, trust, governance, storage, policy, and enforcement.

Summary

The local verification model defines how ALX Protocol verifies Blocks through deterministic computation over protocol data. A conformant implementation can recompute `contentHash` and `blockHash`, validate declared parent lineage, and inspect derivation Graph relationships without relying on a blockchain, token, hosted service, storage provider, registry, index, or external authority.

The Block is the core verification primitive. `content` carries the opaque artifact payload. `parentHashes` carries declared upstream Block identities. `contentHash` provides `content`-only identity. `blockHash` provides lineage-bound Block identity.

This model gives systems and implementers a portable structure for composable digital artifacts. External systems may add storage, signatures, timestamps, schemas, attestations, indexes, policy logic, semantic interpretation, and governance workflows around the Block layer while preserving core protocol identity.

Part IV—Protocol Guarantees and Design Principles

ALX Protocol verifies Block identity, content integrity, declared lineage, and derivation structure through deterministic computation over protocol data. Conforming implementations can independently verify available Blocks and Graphs without relying on blockchains, hosted services, registries, storage providers, or external authorities.

By separating structural verification from application semantics, ALX provides a portable foundation for composable digital artifacts while allowing storage, governance, trust, and interpretation to remain external to the protocol.

Protocol Scope

ALX Protocol defines a minimal structural model for Block identity, lineage binding, graph-based attribution, and local verification. The protocol specifies the required Block fields, the identity relationship between `content`, `parentHashes`, `contentHash`, and `blockHash`, and the structural relationships created when Blocks reference upstream Blocks.
Protocol scope includes:

- deterministic identity generation from canonical protocol data
- `content`-only identity through `contentHash`
- lineage-bound identity through `blockHash`
- declared upstream references through `parentHashes`
- derivation Graph construction from chained Blocks
- structural attribution through `pathCount`
- local verification by conformant implementations

Protocol scope excludes semantic interpretation and external system behavior. ALX Protocol does not determine whether artifact content is true, accurate, complete, compliant, trustworthy, or appropriate for a particular use. These evaluations remain the responsibility of applications and external systems, including governance frameworks, attestation providers, registries, storage systems, institutions, and human reviewers.
This boundary preserves protocol neutrality by separating structural verification from application-specific trust and interpretation. ALX verifies deterministic identity, declared lineage, and structural relationships, while external systems provide storage, governance, signatures, timestamps, policies, legal meaning, trust, and domain-specific judgment.

Minimal Protocol Surface

ALX Protocol defines a minimal protocol surface centered on a single primitive: the Block. Only four required fields participate in protocol identity and verification: content, `parentHashes`, `contentHash`, and `blockHash`.
Together, these fields provide the minimum information required to establish deterministic identity, declared lineage, and independent structural verification. No additional protocol fields are required to create, verify, or compose Blocks into derivation Graphs.
Everything outside this minimal surface remains external to the protocol. Storage networks, databases, blockchains, identity providers, schema frameworks, model providers, governance systems, hosted services, indexes, registries, and application interfaces may store, transmit, interpret, attest to, or govern Blocks without affecting protocol identity or validity.

By limiting the protocol surface to the information required for structural verification, ALX remains independently implementable across diverse environments. Conforming implementations can create, verify, and traverse Blocks using shared protocol rules without depending on a particular storage system, blockchain, execution environment, application framework, or governance model.

Core Protocol Guarantees

ALX Protocol defines four core guarantees at the protocol layer: deterministic identity, parent-bound lineage, traceable attribution, and content opacity. Each guarantee defines a structural property that conforming implementations can verify from protocol data alone.
Together, these guarantees establish deterministic identity, bind declared lineage into Block identity, enable structural attribution through derivation Graphs, and preserve application responsibility for semantic interpretation. They form the foundation for artifact composition and local verification across independent implementations.
The guarantees are intentionally limited to protocol structure. ALX Protocol does not evaluate semantic truth, content quality, model correctness, authorship, legal meaning, economic value, or policy compliance. External systems build upon these guarantees to provide governance, trust, auditing, attribution policies, semantic interpretation, and application-specific behavior.

Deterministic Identity

ALX Protocol defines deterministic identity for each Block. A conformant implementation computes Block identity from canonical protocol data, producing equivalent hash values for equivalent canonical inputs across independent runtimes, programming languages, operating systems, and infrastructure environments.

A Block contains two identity values:

- `contentHash` = `keccak256(canonicalize(content))`
- `blockHash = keccak256(canonicalize({ content, parentHashes }))`
- `contentHash` identifies the artifact payload only. `blockHash` identifies the artifact payload together with declared parent lineage. This separation allows systems to verify `content` integrity and lineage-bound identity as distinct structural properties.
- Deterministic identity enables local verification. A verifier can recompute `contentHash` and `blockHash` from Block data and compare recomputed values with declared values. Matching values confirm that the Block is structurally consistent under ALX Protocol rules.
- Deterministic identity also enables cross-implementation consistency. A Block created in one conformant environment can be verified in another conformant environment through the same canonicalization and hashing rules. Protocol behavior is derived from specification-defined computation rather than a hosted service, proprietary runtime, storage provider, blockchain, or external authority.

Parent-Bound Lineage

ALX Protocol guarantees that every derived Block is cryptographically bound to its declared parent lineage. Parent relationships are incorporated into deterministic Block identity, making the declared dependency structure independently verifiable.

Root Blocks declare no parents, while derived Blocks declare one or more parent Block identities. Because `blockHash` is computed from the canonical content together with the declared parent relationships, any change to the parent set produces a different Block identity.

This guarantee allows digital artifacts to preserve structural dependency relationships across systems. Source materials, intermediate analyses, reviews, policy decisions, agent handoffs, and workflow states can all be represented as declared parent Blocks without requiring the protocol to interpret the meaning of those relationships.

Together, parent relationships form a derivation Graph that represents the declared structural history of an artifact. The protocol guarantees that this lineage is deterministic, portable, and independently verifiable; applications determine what the relationships mean.

Traceable Attribution

ALX Protocol defines traceable attribution as the ability to compute structural relationships within a derivation Graph. As Blocks declare parent relationships, the resulting Graph records how artifacts are structurally connected through successive stages of derivation.

To expose those relationships, ALX computes `pathCount`, the number of distinct directed traversal paths from a selected Trace root to a reachable Block. Because `pathCount` is derived solely from Graph topology, every conforming implementation computes the same structural attribution from the same Block data.

This guarantee is intentionally structural. `pathCount` does not measure semantic importance, authorship, creative contribution, legal ownership, economic value, or policy entitlement. It describes only the connectivity of the Graph. Applications and external systems remain responsible for interpreting what those structural relationships mean.

Traceable attribution supports workflows in which artifacts are repeatedly derived, combined, and reused. Source artifacts may contribute to summaries, summaries to analyses, analyses to syntheses, and syntheses to downstream reports, recommendations, or decisions. Each derivation adds new parent relationships that become part of the computable Graph.

By exposing structural attribution as deterministic protocol data, ALX allows independent systems to inspect how artifacts are connected, determine how many derivation paths link a selected root to a reachable Block, and build higher-level attribution, governance, auditing, or accounting models without changing the protocol itself.

Content Opacity

ALX Protocol defines `content` opacity as a protocol boundary. The protocol verifies artifact structure without interpreting the semantic meaning of `content`. A Block can carry any opaque JSON-serializable payload. The payload may represent a source document, model output, agent handoff, evaluation, decision record, workflow state, tool response, schema object, media

metadata, policy evidence, benchmark result, or other artifact type. Each payload undergoes the same structural process: canonicalization, hash computation, identity verification, and lineage verification.

Content opacity allows ALX to operate across heterogeneous artifact domains. A legal memo, software review, financial analysis, clinical summary, research dataset, compliance assessment, creative output, or machine-generated decision record can participate in the same verification graph without requiring the protocol to understand the domain.

The protocol does not classify `content` type, validate domain schemas, evaluate truth, assess quality, infer intent, enforce policy, or determine compliance. These functions remain external to the Block layer and may be handled by applications, schemas, registries, governance systems, reviewers, or institutional processes. Content opacity preserves a stable verification surface. ALX verifies identity, integrity, lineage, and Graph structure while allowing external systems to define interpretation, presentation, indexing, policy, trust, and domain-specific meaning.

Verification Scope

ALX Protocol defines a clear verification boundary. The protocol verifies determistic structural properties that can be computed from Block data and derivation Graph relationships. Semantic interpretation, legal meaning, economic value, operation behavior, and institutional trust remain the responsibility of applications and external systems.

Structural Verification

The following table summarizes the structural verification process performed by ALX. Each verification stage confirms a deterministic property of the Block, ensuring that its identity, integrity, and declared lineage conform to the protocol specification independently of external trust systems.

Property	Verification Mechanism
Block Identity	Recompute `blockHash` from canonicalized {content, `parentHashes` }
Content Integrity	Recompute `contentHash` from canonicalized `content`
Declared Parent Lineage	Confirm that normalized `parentHashes` are bound into `blockHash` and that each resolved parent reference matches the parent Block's recomputed `blockHash`
Derivation Graph	Traverse declared parent relationships and verify each resolved Block

Graph Completeness	Report the Graph as complete only when every declared parent Block is resolved and verified
Graph Acyclicity	Detect cycles during Graph traversal.
Structural attribution	Compute `pathCount` from verified paths across the resolved Graph
Cross-implementation consistency	Apply shared canonicalization and hashing rules so equivalent protocol data produces equivalent deterministic identities

Table 3. ALX structural verification properties and mechanisms. Structural verification recomputes deterministic Block and content identities, validates declared parent references, traverses the derivation Graph, evaluates completeness and acyclicity, and derives structural attribution from verified paths. A Graph is considered complete only when every declared parent Block is resolved and successfully verified. These mechanisms establish structural integrity and deterministic consistency; they do not establish artifact truth, authorship, quality, or legal validity.

External Trust Domains

The following table summarizes the core fields of an ALX Block and their respective roles in establishing deterministic artifact identity and lineage. Together, these fields define the minimal protocol data model required for structural verification.

Domain	Boundary
Semantic Truth	ALX treats artifact content as opaque data and does not determine whether statements or claims are true.
Content Quality	Structural validity does not imply that content is accurate, complete, useful, or correct.
Model Correctness	Evaluation of AI models and generated outputs is perfomed by applications and model-specific validation systems, not by ALX.
Real-world Authorship	Authorship and identity are established through digital signatures, attestations, credentials, identity systems, or governance mechanism external to ALX.

Storage Availability	parentHashes identify dependencies but do not guarantee that referenced Block are retrievable.
Storage Durability	Persistence is provided by storage networks, repositories, archives, or other external infrastructure.
Policy Compliance	Compliance with organizational policies, regulations, or governance requirements is determined by external policy and governance systems.
Economic Interpretation of `pathCount`	ALX defines pathCount as a structural metric. Attribution, rewards, compensation, or other economic interpretations are assigned by external systems.
Legal Status	Legal rights, ownership, licensing, contractual obligations, and regulatory compliance are determined by applicable legal frameworks, not by the protocol.
Institutional Trust	Trust in organizations, registries, auditors, or human reviewers is established through external governance and institutional processes.
Consensus	ALX does not require blockchain consensus or a global ledger. Anchoring and consensus mechanisms are optional external integrations.

Table 4. ALX protocol boundary and external trust responsibilities. The table distinguishes the deterministic guarantees provided by the ALX Protocol from concerns that must be established by external systems. ALX verifies artifact identity, integrity, and declared lineage, while semantic validity, authorship, governance, legal status, storage guarantees, and other trust properties remain outside the protocol boundary.

Open Protocol Design

ALX Protocol is an open protocol for composable digital artifacts. Open means that the protocol rules, schemas, and canonical test vectors are publicly inspectable and support independent conforming implementations. The protocol defines shared rules for Block structure, canonicalization, hash derivation, parent references, and Graph relationships.

Open protocol design enables independent implementation. Conformant implementations can create Blocks, verify Blocks, and traverse derivation Graphs by following the same protocol rules. Verification does not require coordination through a centralized service, proprietary runtime, hosted platform, storage provider, blockchain network, or application vendor.

The protocol surface remains minimal so external systems can compose around the Block primitive. Storage systems can provide persistence. Attestation systems can provide signed claims. Indexes and registries can provide discovery. Schemas can provide domain-specific structure. Policy systems can provide governance enforcement. Applications can provide semantic interpretation and user-facing workflows.

This design supports ecosystem expansion without changing Block identity. New storage adapters, signing mechanisms, anchoring systems, agent frameworks, indexing systems, application profiles, and governance layers can integrate above the protocol layer while preserving `contentHash`, `blockHash`, and protocol validity.

ALX Protocol therefore provides an independently implementable structure for composable digital artifacts. Blocks provide deterministic identity and declared lineage. External systems provide storage, interpretation, trust signals, policy logic, discovery, and operational behavior.

Summary

ALX Protocol provides a common structural foundation for composable digital artifacts. Blocks establish deterministic identity, declared lineage, and independent structural verification, while applications and external systems provide storage, trust, governance, semantic interpretation, and domain-specific functionality. This separation enables an open, interoperable ecosystem in which independent implementations can compose, verify, and extend digital artifacts without requiring shared infrastructure or centralized coordination.

Part V—Interoperability and Composable Systems

No protocol operates in isolation. Modern digital systems rely on complementary technologies for versioning, content-addressed storage, media provenance, software supply-chain integrity, attestations, trust signatures, shared commitments, observability, and notarization. Each addresses a different aspect of identity, integrity, provenance, trust, or operational visibility.

ALX Protocol provides the structural layer within this broader verification ecosystem. It establishes deterministic identity, declared lineage, and independent verification for digital artifacts while remaining compatible with existing infrastructure rather than replacing it.

This part explains how ALX integrates with complementary systems to create composable verification architectures. External systems continue to provide storage, governance, trust, discovery, and domain-specific functionality, while ALX supplies the shared structural model that allows digital artifacts to remain portable and verifiable across independent environments.

Secure API Build Workflow as an ALX Provenance Graph

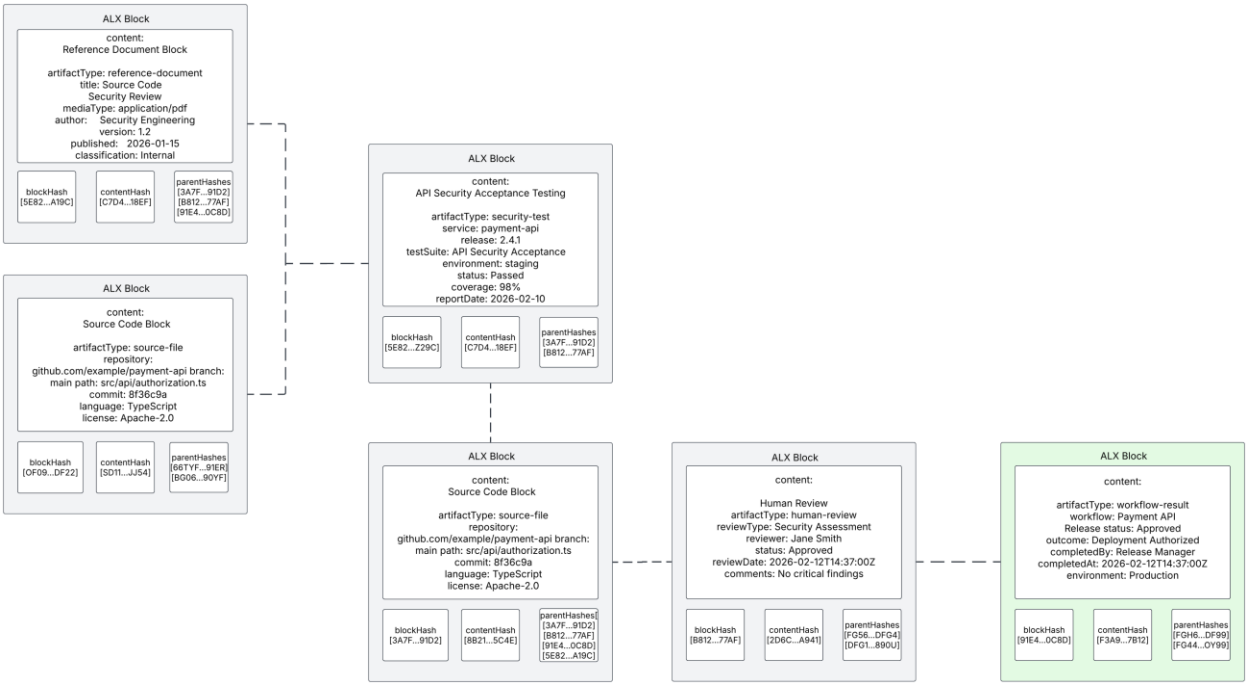

Figure 5. ALX provenance graph for a secure API build workflow. Security documentation, source code, automated security test results, human review, and deployment authorization are represented as deterministically identified ALX Blocks connected through declared parentHashes. Together, these Blocks form a verifiable derivation Graph documenting how the API progresses from implementation and validation to review and authorized deployment, supporting integrity, reproducibility, and auditability throughout the build and release lifecycle.

The Modern Verification Stack

The following table summarizes the modern verification stack, the primary guarantee provided by each layer, and the role ALX plays as a compositional Block layer.

Layer	Representative Systems	Primary Guarantee	ALX Integration
Content Versioning	Git	Repository history, version control, and change tracking	Represents commits, files, or releases as ALX Blocks while preserving deterministic identity and lineage
Content Addressing	IPFS, IPLD	Content-addressable retrieval	References content-addressed objects while maintaining an independent protocol identity
Media Provenance	C2PA	Signed provenance for media assets	References C2PA manifests as parent-linked evidence within an artifact graph
Supply Chain Integrity	SLSA	Build provenance and software supply-chain integrity	Represents build artifacts and attestations as parent-linked evidence within a derivation graph.
Attestation	EIP-712, in-toto	Cryptographically signed assertions	Represents cryptographically signed attestations and while preserving their external trust semantics
Shared Commitment	Blockchains	Append-only commitment and timestamped state	Anchors blockHash values to append-only commitments systems without altering protocol identity or verification
Observability	Logs, traces	Runtime visibility and execution telemetry	Represents logs, traces, and execution records as

			Blocks to enable portable operational lineage
Notarization	Timestamp services	Time-of-existence evidence	Represents timestamp evidence as parent-linked Blocks within a provenance graph
AI Workflow Orchestration	LangGraph, CrewAI, OpenAI Agents SDK, Temporal	Multi-step workflow execution	Workflow inputs, outputs, and intermediate artifacts may be represented as parent-linked Block to create portable, verifiable execution lineage.

Table 5. ALX interoperability within the modern verification stack. ALX complements existing version control, content-addressing, provenance, attestation, storage, and commitment systems by providing deterministic artifact identity and parent-bound lineage. Rather than replacing these technologies, ALX enables them to participate in a unified, portable provenance graph through interoperable Block representations and references.

Positioning ALX Within the Verification Stack

ALX Protocol operates at the structural layer of the modern verification stack. Rather than replacing versioning systems, content-addressed storage, provenance standards, attestations, or commitment systems, ALX provides a common representation for digital artifacts that can interoperate across them.

Each Block represents a digital artifact with deterministic identity and declared lineage. These structural properties allow artifacts to be linked, composed, and independently verified regardless of the storage system, application, execution environment, or trust framework in which they originate.

Because the Block model is compositional, external systems can integrate without altering protocol identity. Git commits, IPFS objects, C2PA manifests, SLSA attestations, signatures, timestamps, blockchain commitments, and operational records may be referenced by, represented as, or associated with Blocks while preserving deterministic identity and declared lineage.

This separation allows every layer to retain its native responsibilities. Existing systems continue to provide storage, provenance, trust, governance, discovery, and operational visibility, while ALX supplies the portable structural model that enables digital artifacts to move between those systems without losing verifiable identity or lineage.

Integration with Existing Systems

ALX Protocol interoperates with existing verification, provenance, storage, and trust systems through the Block model. Rather than replacing established technologies, the protocol provides

a shared structural representation that enables digital artifacts to retain deterministic identity, declared lineage, and independent verification across system boundaries.

Git repositories, content-addressed storage, media provenance standards, supply-chain attestations, signatures, timestamps, blockchains, and other complementary systems continue to provide their native guarantees. ALX provides the portable artifact layer that allows these systems to compose within a common derivation Graph while preserving their existing roles and formats.

By separating structural verification from storage, governance, trust, and semantic interpretation, ALX establishes a common foundation for interoperable, composable digital artifacts across independent applications, organizations, and verification ecosystems.

Interoperability Domains

Content Versioning

Content versioning systems record the evolution of repositories, files, and software projects. Git, for example, provides repository history, commits, branches, diffs, and authorship metadata, enabling coordinated software development and reproducible project histories.
ALX Protocol complements content versioning by allowing versioning records to participate in artifact lineage. A Block may reference repository objects such as commits, tags, releases, or snapshots, or represent those records directly as Block content. When represented as Blocks, versioning records become part of the same derivation Graph as AI outputs, evaluations, workflow states, policy decisions, and other digital artifacts.
This integration extends traceability beyond repository boundaries. Repository history can be linked with artifacts produced by models, agents, APIs, workflow engines, and external systems, allowing software development and AI workflows to participate in a common structural Graph.
Git continues to provide repository semantics and version history. ALX provides deterministic artifact identity, declared lineage, and structural verification that remain portable across environments where repository history alone is insufficient.

Content Addressing and Distributed Storage

Content-addressed storage identifies data through content-derived identifiers rather than storage location. Systems such as IPFS and IPLD provide portable retrieval across distributed storage environments while allowing identical content to resolve to the same address.
ALX complements content-addressed storage by distinguishing payload identity from lineage-bound identity. Content-addressed objects may be referenced within Blocks or represented directly as Block content, allowing retrieved documents, datasets, media assets, and other stored objects to participate in derivation Graphs.
This separation enables distributed storage systems to provide persistence and retrieval while ALX provides deterministic identity, declared lineage, and structural verification. Storage infrastructure and artifact lineage therefore remain independent concerns.

ALX does not replace distributed storage. Storage systems remain responsible for persistence and availability, while ALX provides the portable structural model that allows stored artifacts to participate in interoperable verification workflows.

Media Provenance

Media provenance systems record the origin, transformation history, and authenticity claims associated with digital media. Standards such as C2PA provide signed provenance metadata for images, video, audio, and other media assets, enabling consumers to inspect how media was created or modified.

ALX Protocol complements media provenance by allowing media assets, provenance manifests, edit histories, and verification records to participate in a common artifact Graph. Media records may be referenced by Blocks or represented directly as Block content, allowing them to compose with prompts, model outputs, evaluations, agent handoffs, policy decisions, and other workflow artifacts.

This integration extends provenance beyond individual media assets. Media records can participate in broader derivation Graphs that connect creative workflows, AI-generated content, review processes, and downstream applications while preserving deterministic identity and declared lineage.

ALX does not determine media authenticity, authorship, copyright ownership, consent, or the validity of provenance claims. Media provenance systems continue to provide those guarantees, while ALX provides the structural layer that links media artifacts into portable, independently verifiable workflows.

Supply Chain and Build Integrity

Supply chain and build integrity systems provide verifiable evidence describing how software, models, datasets, and other digital artifacts are produced. Frameworks such as SLSA document build pipelines, dependencies, provenance, and generated outputs, helping organizations assess the integrity and reproducibility of development processes.

ALX Protocol complements supply chain systems by allowing build artifacts, provenance records, dependency information, model checkpoints, dataset versions, and release metadata to participate in a common artifact Graph. Supply chain records may be referenced by Blocks or represented directly as Block content, allowing them to compose with source materials, AI workflows, evaluations, deployment records, and downstream artifacts.

This integration extends build provenance beyond software pipelines. Supply chain evidence can be linked with the broader lifecycle of digital artifacts, connecting development, AI generation, review, deployment, and operational workflows through deterministic identity and declared lineage.

ALX does not determine whether a build is secure, reproducible, compliant, approved, or free of vulnerabilities. Supply chain frameworks, security tools, attestations, and governance systems continue to provide those guarantees, while ALX provides the structural layer that links build evidence into portable, independently verifiable derivation Graphs.

Attestation and Trust Systems

Attestation and trust systems provide signed assertions about digital artifacts, actors, execution environments, compliance, approvals, and validation events. Technologies such as EIP-712, in-toto, digital signatures, and verifiable credentials enable external parties to make verifiable claims about artifacts and the processes that produced them.

ALX Protocol complements attestation and trust systems by allowing attestations, credentials, approval records, validation results, and other trust artifacts to participate in a common derivation Graph. These records may be referenced by Blocks or represented directly as Block content, allowing trust information to compose with source materials, AI outputs, agent workflows, evaluations, and downstream artifacts.

This integration preserves a clear separation between structure and trust. ALX provides deterministic identity and declared lineage, while attestation systems provide cryptographic claims about authenticity, approval, compliance, or authorship. Together, they allow structural relationships and trust assertions to travel with digital artifacts across independent systems.

ALX does not determine whether an attestation is truthful, authoritative, legally sufficient, or issued by a trusted party. Trust frameworks, signature verification systems, credential providers, governance processes, and application policies continue to evaluate those claims, while ALX provides the structural layer that links them into portable, independently verifiable artifact Graphs.

Shared Commitment Systems

Shared commitment systems provide durable, publicly verifiable records of submitted data, state transitions, timestamps, and existence proofs. Blockchains, transparency logs, append-only ledgers, and commitment registries enable multiple parties to independently reference and verify shared records over time.

ALX Protocol complements these systems by allowing commitments, transaction records, anchor receipts, Merkle proofs, and registry entries to participate in a common derivation Graph. Commitment records may be referenced by Blocks or represented directly as Block content, allowing public commitments to compose with source materials, AI workflows, evaluations, attestations, and downstream artifacts.

This integration separates structural identity from shared commitment. ALX establishes deterministic identity and declared lineage independently of any external infrastructure, while shared commitment systems provide optional public anchoring, timestamping, and multi-party coordination. Together, they allow independently verifiable artifacts to be associated with durable public records without changing protocol identity.

ALX does not require a blockchain, token, ledger, registry, or consensus mechanism for protocol validity. Shared commitment systems remain optional infrastructure that strengthens coordination and public verifiability, while ALX provides the portable structural layer that links those commitments into interoperable derivation Graphs.

Observability Systems

Observability systems provide runtime visibility into system behavior through logs, traces, metrics, events, spans, execution records, and operational telemetry. These records help developers and operators understand how applications, agents, models, APIs, and workflows execute across distributed environments.

ALX Protocol complements observability systems by allowing runtime events, execution records, tool outputs, agent actions, model responses, and telemetry data to participate in a common derivation Graph. Observability records may be referenced by Blocks or represented directly as Block content, allowing operational history to compose with source materials, AI outputs, evaluations, policy decisions, and downstream artifacts.

This integration extends observability beyond runtime diagnostics. Execution history can remain associated with the digital artifacts it helped produce, enabling workflows to preserve verifiable operational context alongside deterministic identity and declared lineage across system boundaries.

ALX does not determine whether telemetry is complete, whether logs are accurate, whether traces capture every execution step, or whether operational events satisfy policy requirements. Observability platforms, monitoring systems, and governance processes continue to provide those guarantees, while ALX provides the structural layer that links operational records into portable, independently verifiable artifact Graphs.

Notarization Systems

Notarization systems provide evidence that a digital artifact, identifier, claim, or record existed, was submitted, or was observed at a particular point in time. Timestamp services, certification workflows, audit systems, organizational notaries, and registry services establish external records that support accountability, auditing, and legal or operational processes.

ALX Protocol complements notarization systems by allowing timestamp records, submission receipts, audit entries, registry records, certificates, and approval events to participate in a common derivation Graph. These records may be referenced by Blocks or represented directly as Block content, allowing notarization events to compose with source materials, AI workflows, evaluations, attestations, and downstream artifacts.

This integration preserves the relationship between an artifact and the external records that document its existence or approval. Notarization evidence can travel alongside deterministic identity and declared lineage, allowing independent systems to verify structural relationships while retaining external proof of time, submission, or certification.

ALX does not determine whether a timestamp is authoritative, a notary is trusted, a certificate is valid, or an approval satisfies legal or organizational requirements. Notarization services, legal frameworks, and governance processes continue to provide those guarantees, while ALX provides the structural layer that links notarization records into portable, independently verifiable artifact Graphs.

Composable Workflows

Composable workflows emerge when digital artifacts carry deterministic identity and declared lineage across system boundaries. Rather than existing as isolated outputs within individual applications, artifacts become reusable building blocks that can be referenced, verified, extended, and recombined throughout a workflow.

ALX Protocol provides the structural foundation for this composition. Blocks establish portable artifact identity, parent relationships capture declared dependencies, and derivation Graphs preserve how artifacts relate as workflows evolve across tools, organizations, and execution environments.

The following examples illustrate how this model supports multi-party AI workflows, reproducible research, source-grounded generation, contribution analysis, and machine-consumable context. Together, they demonstrate how deterministic artifact composition enables interoperable workflows that extend beyond any single application or platform.

Multi-Party AI Workflows Using Artifact Graphs

AI workflows increasingly span multiple systems, execution environments, model providers, data sources, agents, and organizational boundaries. A single workflow may involve retrieval systems, orchestration frameworks, model calls, tool invocations, policy checks, human review, storage platforms, and downstream applications.

Each stage produces digital artifacts that influence subsequent decisions. A credit assessment may depend on customer records, policy rules, scoring models, analyst reviews, and approval decisions. A trading workflow may combine market data, news analysis, technical indicators, compliance checks, execution reports, and audit records. A legal workflow may build upon source documents, extracted claims, draft revisions, review feedback, and final filings.

ALX represents these workflows as derivation Graphs. Source materials, intermediate results, agent handoffs, evaluations, decisions, and final outputs can each be represented as Blocks connected through declared parent relationships. Rather than treating workflow state as isolated application data, ALX preserves it as a portable, verifiable artifact history.

This model supports workflows in which no single platform owns the complete context. Independent systems can contribute artifacts, extend existing work, and verify declared dependencies without requiring shared infrastructure or centralized control. As artifacts move between applications, organizations, and AI agents, deterministic identity and declared lineage remain intact.

By representing workflow history as composable protocol data, ALX enables multi-party AI systems to exchange artifacts without losing traceability. Independent implementations can inspect, verify, and extend artifact Graphs while allowing each participant to retain its own execution environment, governance model, and application semantics.

Reproducible AI Research

AI research depends on reproducible digital artifacts. Benchmark studies often combine datasets, prompt sets, evaluation scripts, model checkpoints, model cards, intermediate

outputs, aggregated results, and published conclusions. As these components evolve over time, reproducing prior work becomes increasingly difficult unless their relationships can be preserved.

ALX Protocol provides a structural foundation for reproducible research by representing each research artifact as a Block with deterministic identity and declared lineage. Datasets, prompts, evaluation scripts, intermediate results, analyses, and published findings become interconnected artifacts within a common derivation Graph.

This representation allows researchers and independent reviewers to inspect how published results were produced, identify the declared dependencies of an experiment, and determine whether the required artifacts remain available. Rather than relying solely on repository history or documentation, the complete research workflow can be represented as portable, machine-verifiable artifact structure.

Reproducibility therefore becomes a property of the artifact Graph rather than a collection of disconnected files and records. As research is revised, extended, or independently reproduced, new Blocks preserve both the original artifact history and the lineage of subsequent work.

When the required Blocks are available, conforming implementations can independently verify the declared structure of the research workflow. Even when some artifacts become unavailable, deterministic Block identities preserve stable references to the expected artifacts and their declared relationships, supporting future verification as data becomes available.

Source-Grounded Derivation

Retrieval-augmented generation systems, copilots, and autonomous agents increasingly depend on approved source material. Generated outputs may draw upon internal documents, regulatory filings, knowledge bases, policy manuals, customer records, market data, or other controlled sources, making transparent source relationships essential for verification and governance.

ALX Protocol represents source grounding as declared structural lineage. Source documents, retrieved passages, policy definitions, datasets, and other supporting materials can each be represented as Blocks. Derived outputs preserve these relationships by declaring their upstream dependencies within the derivation Graph.

This representation makes source grounding independently inspectable. Applications and governance systems can determine whether outputs declare dependencies on approved source artifacts, identify where external or unapproved sources have entered a workflow, and evaluate grounding policies using a common structural model rather than application-specific metadata.

ALX does not determine whether a generated output faithfully represents its sources or whether a cited source is factually correct. Those evaluations remain the responsibility of applications, governance frameworks, and human reviewers. ALX provides the structural layer that preserves declared source relationships as portable, independently verifiable artifact lineage.

Quantifiable Contribution Structures

Modern AI workflows increasingly involve contributions from multiple people, organizations, models, agents, datasets, and software systems. Research collaborations, open-source

projects, enterprise workflows, and multi-vendor platforms often produce artifacts whose outcomes depend on many upstream contributors.

ALX Protocol provides a structural basis for analyzing these relationships through `pathCount`, the number of distinct directed traversal paths from a selected Trace root to a reachable Block. Because `pathCount` is derived solely from Graph topology, every conforming implementation computes the same structural attribution from the same artifact Graph.

This metric describes structural connectivity, not semantic or economic value. `pathCount` does not determine authorship, creative contribution, legal ownership, compensation, ranking, or policy entitlement. It quantifies only how artifacts are connected through declared lineage.

Applications and external systems can build upon this structural foundation to perform contribution analysis, royalty allocation, dependency accounting, governance review, provenance scoring, or other domain-specific attribution models. ALX supplies the deterministic structural metric; interpretation remains the responsibility of applications, institutions, and policy frameworks

From Artifact Composition to Machine-Consumable Context

Digital artifacts become more valuable when their relationships can be reused across workflows rather than reconstructed for each execution. By preserving deterministic identity and declared lineage, ALX enables applications to assemble machine-consumable context from previously verified artifacts instead of relying solely on transient execution state.

Applications can traverse derivation Graphs to construct context from source materials, intermediate results, agent handoffs, evaluations, policy decisions, workflow states, and other related artifacts. This context remains an application-layer interpretation; the protocol continues to define only deterministic identity, declared lineage, and structural relationships.

This model allows context that would otherwise exist only within prompts, memory, logs, traces, or application state to become portable artifact data. As these records are represented by Blocks, they can be referenced, verified, reused, and extended across independent systems while preserving their structural relationships.

Applications can therefore continue work from verifiable prior state rather than rebuilding context from scratch. Models can consume artifacts together with their declared dependencies, agents can extend existing workflows, applications can reuse verified outputs, and auditors can inspect artifact histories without requiring access to the original execution environment.

Example: Verifiable Trading Pipeline

The following example illustrates how ALX represents an AI-driven trading workflow as a derivation Graph. Data sources, analytical stages, agent handoffs, compliance decisions, execution records, and audit artifacts are each represented as Blocks connected through declared parent relationships. While the protocol treats every Block as opaque content, applications assign domain-specific meaning such as datasets, analyses, signals, trades, or audit records.

The workflow begins with independent market and news data that feed specialized analysis agents. Sentiment analysis and technical analysis produce intermediate artifacts that are

combined into a trading signal. That signal is evaluated alongside the current portfolio state to produce a proposed trade, which then passes through compliance review before execution. The resulting execution report updates portfolio state, supports post-trade risk analysis, and contributes to a complete audit trail. An independent back testing branch demonstrates how historical validation can coexist with the live execution workflow while preserving separate lineage.

Because every stage is represented as a Block, the complete decision history becomes structurally verifiable. Systems can inspect declared dependencies, determine which artifacts contributed to a trading decision, verify the integrity of available workflow records, and distinguish live execution from offline analysis without relying on the originating application or execution environment.

This example illustrates how ALX scales from individual artifacts to complex institutional workflows. The same Block model represents data ingestion, AI analysis, human or automated review, execution, governance, auditing, and historical validation within a single composable artifact Graph.

Summary

ALX Protocol enables digital artifacts to move across applications, organizations, and verification systems without losing deterministic identity or declared lineage. By representing artifacts as composable Blocks connected through derivation Graphs, the protocol provides a common structural foundation for interoperable workflows.

The examples in this part demonstrate how that foundation extends beyond individual applications. Multi-party AI systems, reproducible research, source-grounded generation, contribution analysis, and machine-consumable context all become portable, inspectable, and independently verifiable through a shared artifact model.

ALX defines the structural layer for artifact composition. Applications and complementary systems continue to provide storage, governance, trust, discovery, semantic interpretation, and domain-specific functionality. Together, these layers enable interoperable ecosystems built upon a common protocol for digital artifacts.

Part VI—Protocol Adoption and Extension

ALX Protocol defines the Block primitive and the deterministic rules for Block identity, declared lineage, and derivation Graph structure. Human users, applications, SDKs, middleware, agents, orchestration systems, CI/CD pipelines, and governance workflows can create, exchange, and manage Blocks around digital artifacts that benefit from durable identity, declared lineage, and independent verification.

Block creation is implementation-agnostic. Conforming implementations may create Blocks manually, programmatically, or automatically within application workflows. Regardless of how a Block is created, protocol validity is determined solely by the requirements defined in the specification, schemas, and canonical test vectors.

Once created, Blocks become reusable structural artifacts that other systems can reference, verify, compose, and extend through declared parent relationships. External systems may associate storage references, attestations, timestamps, schemas, indexes, registries, anchors, governance metadata, or application-specific context with a Block without changing its identity or protocol validity.

Creating Blocks

A Block should be created when a digital artifact requires a durable, portable representation with deterministic identity and declared lineage. The deciding factor is not the artifact type, but whether downstream systems may need to reference, verify, reuse, inspect, audit, or extend the artifact beyond its original execution context.

Block creation is generally appropriate for artifacts that:

- contribute to downstream workflows or decisions
- serve as evidence, references, or supporting material
- move between agents, applications, systems, or organizations
- undergo review, approval, validation, or governance
- are published, exported, archived, or externally shared
- support reproducibility, auditing, or dispute resolution
- provide persistent application, organizational, or institutional context

Not every artifact requires a Block. Transient execution state, temporary variables, ephemeral logs, or short-lived operational events may never need durable identity or lineage. Implementers should create Blocks for artifacts whose value extends beyond the immediate execution in which they were produced.

This principle allows ALX to remain application-agnostic. Any digital artifact—from a prompt, dataset, and model output to a workflow decision, software release, or regulatory record—can be represented as a Block whenever durable identity, declared lineage, and future verification provide lasting value.

Creation Modes

ALX Protocol supports multiple Block creation modes. These modes describe how Blocks are created and introduced into workflows without affecting protocol behavior. Regardless of how a Block is created, deterministic identity, declared lineage, verification rules, and protocol validity remain unchanged.

Manual Creation

Manual creation occurs when a user explicitly creates a Block through a user interface, command-line tool, document workflow, review system, publication process, or other interactive application.

Manual creation is appropriate when an artifact is intentionally selected for long-term reference, review, publication, governance, or archival. Examples include approved policy decisions, cited

source documents, reviewed drafts, signed reports, published research, regulatory submissions, and finalized records.
Manual creation allows people to determine which artifacts become durable, reusable components within a derivation Graph.

Programmatic Creation

Programmatic creation occurs when an application, service, SDK, API, or backend process creates a Block during normal execution.
Programmatic creation is appropriate when software produces artifacts that may be referenced, reused, verified, or extended by downstream systems. Examples include model outputs, tool responses, workflow results, evaluation records, repository references, build artifacts, compliance checks, and generated reports.
Programmatic creation enables applications to preserve artifact identity and lineage automatically as part of existing workflows, without requiring manual intervention.

Automatic Creation

Automatic creation occurs when middleware, agent frameworks, orchestration platforms, retrieval pipelines, CI/CD systems, or workflow engines create Blocks at predefined points during execution.
Automatic creation is appropriate when workflow transitions, handoffs, transformations, approvals, or state changes should be preserved as durable artifacts without requiring explicit user or application logic. Examples include agent handoffs, retrieval bundles, prompt-output pairs, policy evaluations, execution traces, deployment decisions, and audit events.
Automatic creation allows systems to capture workflow history consistently as it occurs, ensuring that significant execution events become reusable, verifiable artifacts while remaining transparent to end users and applications.

SDK-Mediated Block Creation

An SDK allows Block creation to become a native part of application and workflow execution. Applications submit a digital artifact and its declared dependencies, while the SDK applies the protocol rules to produce a valid Block. This includes content normalization, canonicalization, deterministic hash generation, and Block assembly in accordance with the ALX specification.
SDK-mediated creation enables applications to represent artifacts at the point they are produced. Retrieval systems can create Blocks for source bundles, model gateways for prompt–response pairs, agent frameworks for workflow handoffs, CI/CD pipelines for build records, and governance systems for review decisions or policy outcomes.

A typical SDK-mediated workflow includes:

- receive or observe a digital artifact
- normalize the artifact into protocol content
- collect declared parent Block references
- apply canonicalization and deterministic hash generation

- emit a valid Block
- optionally associate storage references, attestations, timestamps, indexes, registries, anchors, or application metadata

SDKs simplify integration without changing protocol behavior. Regardless of implementation language or runtime environment, every SDK applies the same canonicalization, hashing, and validation rules defined by the ALX specification. Blocks created through an SDK remain interoperable with every other conforming implementation.
Beyond Block creation, SDKs may provide optional developer tooling such as storage adapters, Graph traversal utilities, verification helpers, registry integration, attestation workflows, and anchoring support. These capabilities improve developer experience while remaining external to Block identity and protocol validity.

External Attachments and Operations

External attachments allow applications and infrastructure to associate additional context, evidence, storage references, or operational metadata with a Block without changing its deterministic identity. These attachments operate entirely above the protocol layer and remain external to Block validity.

External attachments may include:

- storage references
- content-addressed identifiers such as CID or `cidHash`
- attestations and signatures
- timestamps
- schemas and `content` envelopes
- indexes and registry entries
- policy metadata
- verification receipts
- application metadata
- governance records
- hosted discovery records

These attachments extend how Blocks are stored, discovered, interpreted, governed, and trusted. For example, storage references support retrieval, attestations provide signed claims, timestamps establish external time evidence, schemas describe application-specific structure, registries enable discovery, and verification receipts record the outcome of completed verification operations.
Because external attachments do not participate in Block identity computation, they can be added, updated, or removed without affecting deterministic identity, declared lineage, or protocol validity. This separation allows applications to evolve independently while preserving stable Block identities across implementations.

Applications can therefore build rich operational ecosystems around Blocks—including storage, trust, governance, indexing, access control, and user-facing experiences—without changing the underlying protocol object or affecting interoperability.

Anchoring, Indexes, and Registries

Anchoring, indexes, and registries provide optional infrastructure for publishing, discovering, coordinating, and referencing Blocks beyond the protocol layer. These systems improve operational usability without affecting Block identity, declared lineage, or protocol validity.
Anchoring allows one or more blockHash values to be committed to an external system such as a blockchain, transparency log, or other shared commitment mechanism. Anchors can provide existence-time evidence, publication records, or shared reference points for groups of Blocks while remaining external to protocol verification.
Indexes organize Blocks and derivation Graphs to support search, retrieval, filtering, traversal, and application-specific discovery. Registries provide managed publication surfaces for Blocks, Graph roots, schemas, extensions, verification receipts, anchors, application profiles, and other protocol-related resources. Both layers improve accessibility and coordination without becoming part of Block identity.
These infrastructure layers complement the protocol rather than extending it. Applications may publish Blocks to registries, index Graphs for efficient discovery, or anchor selected Block identities for public coordination. Regardless of these operational services, any conforming implementation can verify a Block directly from protocol data. ALX therefore separates deterministic protocol validity from optional infrastructure for discovery, publication, coordination, and external commitment.

Governance and Policy Layers

Governance and policy layers operate above the Block primitive. They define how organizations apply access control, retention policies, review procedures, compliance requirements, approval workflows, audit processes, publication rules, dispute resolution, and institutional trust over digital artifacts.
ALX Protocol provides the structural foundation that governance systems evaluate. Applications and institutions can inspect Block identity, declared lineage, derivation Graph relationships, attestations, verification receipts, timestamps, schemas, registries, and other associated records before applying domain-specific policies and operational decisions.
Protocol validity and policy compliance remain independent. A Block may be structurally valid while failing an organizational policy, regulatory requirement, legal obligation, or institutional review. Likewise, approving or rejecting a Block through governance processes does not change its deterministic identity or declared lineage.
This separation allows different organizations, industries, and jurisdictions to apply independent governance models over the same protocol data. One implementation may require signed attestations, another approved source artifacts, another closed-world verification, while others permit open-world verification with declared external boundaries. ALX provides the common

structural foundation; governance systems provide authorization, compliance, trust, legal interpretation, and operational enforcement.

Economic Coordination

XNDR is a token designed to support ecosystem participation, approved contribution programs, incentives, and other economic coordination activities. The economic model, governance, and legal framework for XNDR are described separately in the XNDR Economy Whitepaper.
ALX Protocol does not depend on XNDR for protocol operation or conformance. Deterministic Block identity, declared lineage, local verification, and derivation Graph traversal remain independently implementable using only the ALX Protocol specification, schemas, and canonical test vectors.
Applications and ecosystem programs may use verified Block data as inputs to economic coordination, contribution recognition, incentive models, or other application-layer decisions. These economic mechanisms operate above the protocol layer and do not affect Block identity, protocol validity, or interoperability.

Summary

ALX Protocol provides an implementation-agnostic model for creating and managing composable digital artifacts. Human users, applications, SDKs, middleware, agents, orchestration systems, CI/CD pipelines, and governance workflows can create Blocks whenever durable identity, declared lineage, and future verification provide lasting value.
External infrastructure—including storage systems, attestations, timestamps, schemas, indexes, registries, anchors, verification receipts, governance frameworks, and application metadata—can extend Block usability without changing deterministic Block identity or protocol validity.
By separating protocol structure from infrastructure, governance, and economic participation, ALX enables independent implementations to create, verify, exchange, and extend composable digital artifacts while preserving interoperability across applications, organizations, and ecosystems.

Part VII—Protocol Specification and Conformance

ALX Protocol supports independent implementation through shared canonicalization rules, schemas, hash derivation, and conformance requirements. Equivalent protocol inputs produce equivalent Block identities and verification results across conforming implementations.
The following sections summarize the canonicalization rules, hash function and encoding, Block schema, parent hash requirements, Graph validation modes, Attribution Trace, verification receipts, extension mechanisms, compatibility expectations, and conformance requirements that define interoperable implementations.

Canonicalization Rules

Canonicalization converts protocol data into a deterministic byte representation before hash computation. Conforming implementations MUST produce equivalent canonical byte sequences for equivalent protocol data.

ALX applies canonicalization to:

- content when computing contentHash
- {content, parentHashes} when computing blockHash

Canonicalization rules include:

- object keys are sorted lexicographically
- array ordering is preserved
- top-level undefined values and undefined values within arrays canonicalize to null
- object properties with undefined values are omitted
- strings are encoded as UTF-8
- numbers are represented according to the canonical JSON encoding rules
- boolean values remain unchanged
- null values remain unchanged

Equivalent protocol data MUST produce equivalent canonical byte sequences across all conforming implementations.

Hash Function and Encoding

ALX Protocol uses Keccak-256 for deterministic protocol hash derivation. Hash inputs are the canonical byte representations produced by the ALX canonicalization rules.

- contentHash is computed from canonicalized content
- contentHash = keccak256(canonicalize(content))
- blockHash is computed from canonicalized { content, parentHashes }:
- blockHash = keccak256(canonicalize({ content, parentHashes }))

Conforming implementations MUST encode hash values as:

- 0x prefix
- lowercase hexadecimal characters
- 64 hexadecimal characters following the prefix
- a 32-byte digest

This encoding provides a consistent representation for `contentHash`, `blockHash`, and `parentHashes` across conforming implementations.

Matching hash values confirm deterministic identity under the ALX Protocol rules. Hash derivation verifies protocol structure only and does not determine semantic truth, content quality, authorship, storage availability, policy compliance, or legal status.

Block Schema

The Block schema defines four required protocol fields—`blockHash, contentHash`, `parentHashes`, and `content`—and permits the optional `protocolVersion` field defined by protocol version 1.

```
{
  "blockHash": "0x...",
  "contentHash": "0x...",
  "parentHashes": ["0x..."],
  "content": {}
}
```

`blockHash` is the deterministic identity of the Block. It is derived from canonicalized `{content, parentHashes}`. Any change to the content or declared parent set produces a different `blockHash`.
`contentHash` is the deterministic identity of the Block content. It is derived from canonicalized `content`. Any change to the content produces a different `contentHash`.
`parentHashes` is the normalized list of declared upstream Block identities. Each entry MUST conform to the protocol hash format. A root Block contains an empty `parentHashes` list.
`content` is the opaque JSON-serializable artifact payload. ALX Protocol does not inspect, classify, or interpret its semantic meaning.
The Block schema validates protocol structure, including required fields, the optional `protocolVersion` field, field names, hash formats, array constraints, and permitted properties. Schema validation alone does not verify Block identity. Full verification requires deterministic recomputation of `contentHash` and `blockHash` from canonical protocol data.

Parent Hash Rules

`parentHashes` defines the declared upstream Block references for a Block. Each entry identifies an upstream Block by `blockHash`.
A root Block MUST contain an empty `parentHashes` list. A derived Block MUST contain one or more parent references. Before `blockHash` is computed, conforming implementations MUST normalize `parentHashes` by:

- converting valid hash values to lowercase
- discarding invalid hash values
- removing duplicate entries
- Sorting the retained entries lexicographically

The normalized `parentHashes` list MUST NOT contain more than 256 entries.
Because the normalized `parentHashes` list participates in `blockHash` derivation, any change to the retained parent set produces a different `blockHash`.

Graph Validation Modes

`parentHashes` defines the declared upstream Block references for a Block. Each entry identifies an upstream Block by its blockHash.

A root Block contains an empty `parentHashes` list. A derived Block contains one or more parent references. Each parent reference declares a dependency relationship between the derived Block and an upstream Block.
Before `blockHash` is computed, `parentHashes` is normalized by:

- lowercasing valid hash values
- discarding invalid hash values
- removing duplicate entries
- sorting the retained entries lexicographically

The normalized parentHashes list MUST NOT contain more than 256 entries.
Because the normalized parentHashes list participates in blockHash derivation, any change to the retained parent set produces a different blockHash.
Schema validation confirms the structural representation of parentHashes. Parent resolution and derivation Graph verification are performed separately according to the selected verification mode.

Attribution Trace

An Attribution Trace is a computed structural view of a derivation Graph. Starting from a selected root Block, the trace records Graph topology, resolved nodes, directed edges, traversal depth, unresolved parents, declared external boundaries, cycle status, and structural attribution metrics.

An Attribution Trace can include:

- `root` — the blockHash of the Block where tracing begins
- `validationMode` — the Graph validation mode used during traversal
- `complete` — whether all required lineage was resolved under the selected validation mode
- `nodes` — resolved Blocks included in the trace
- `edges` — directed parent-reference relationships between Blocks
- `leaves` — resolved nodes with no resolved parents
- `missingParents` — unresolved parent hashes not declared as external boundaries
- `externalParents` — unresolved parent hashes declared as external boundaries
- `maxDepth` — longest path length from the root Block to any traced node
- `cycle` — whether traversal detected a cycle

Each traced node may include derived Graph metrics such as `minDepth, maxDepth, pathCount, parentCount,` and `childCount. pathCount` records the number of distinct directed traversal paths from the selected trace root to the node.

An Attribution Trace is computed from the resolved derivation Graph and supports lineage inspection, contribution analysis, reproducibility, auditing, visualization, and other application-layer workflows. It does not establish semantic truth, content quality, authorship, policy compliance, legal interpretation, or economic value.

The Attribution Trace schema validates structural representation only. The correctness of a computed trace depends on the available Block data, the selected validation mode, the declared verification context, and the implementation performing the traversal.

Verification Receipts

A verification receipt is a portable record of a completed ALX verification operation. Recorded details include the verification context, verifier identity, graph root, validation mode, resolved graph size, unresolved boundaries, outcome, and any errors or warnings.

A verification receipt can include:

- `receiptVersion` — the version of the receipt format
- `protocolVersion` —ALX Protocol version used during verification
- `verifiedAt` — timestamp when verification completed
- `verifier` — name and version of the implementation that produced the receipt
- `root` — `blockHash` of the verified graph root
- `validationMode` — verification mode used during traversal
- ok — whether verification succeeded under the selected validation mode
- `complete` — true only when every required parent resolves under the selected validation mode
- `blockCount` — number of resolved Block
- `edgeCount` — number of traversed parent-reference edges
- `externalParents` — declared as external parent references
- `missingParents` — unresolved parent references not declared as external boundaries
- `errors` — structured verification errors
- `warnings` — non-fatal verification warnings
- `traceHash` — optional hash of the associated Attribution Trace

Verification receipts support auditing, reproducibility, governance, and application integration. They may be attached to Blocks, published through registries, stored with audit records, or exchanged as evidence that a specific verification operation was performed.
A verification receipt is external to the Block. Creating, modifying, storing, publishing, or deleting a receipt does not affect the contentHash, blockHash, declared lineage, or protocol validity of the Block.
A verification receipt records the outcome of a verification operation. It does not establish semantic truth, content quality, authorship, legal status, policy compliance, storage durability, or institutional trust. Those determinations remain the responsibility of applications, governance frameworks, and other external systems.

Extensions and Compatibility

ALX Protocol defines a stable core protocol and permits optional extensions above or alongside that core. Extensions may introduce schemas, content profiles, verification utilities, indexing behavior, registry profiles, attestation formats, anchoring patterns, workflow adapters, or domain-specific conventions without modifying the core protocol.

Protocol version 1 defines deterministic Block identity through canonicalized content and normalized parentHashes, producing contentHash and blockHash. Optional extensions MUST NOT alter these core identity rules.

Each extension SHOULD declare its lifecycle status, compatible protocol versions, required schemas, conformance expectations, and associated test vectors. Lifecycle status may include Draft, Provisional, Stable, Deprecated, or Withdrawn.

Applications may adopt extensions for typed content envelopes, domain-specific schemas, registry profiles, attestation workflows, SDK adapters, Graph export formats, policy profiles, or interoperability packages. These capabilities extend ecosystem functionality while preserving the minimal protocol surface.

Core protocol validity is independent of optional extensions. A Block that conforms to the core protocol remains valid even when an extension is unavailable or unsupported. Applications may require specific extensions for their own workflows, but those requirements remain application-layer policy unless incorporated into the core protocol.

This extension model allows the ALX ecosystem to evolve without fragmenting interoperability. New capabilities can be introduced while preserving deterministic identity, declared lineage, local verification, and cross-implementation compatibility for conforming Blocks.

Conformance Requirements

The canonical specification defines the normative requirements for Block structure, canonicalization, hash derivation, parent handling, local verification, and derivation Graph traversal.

Core conformance requires an implementation to:

- accept and validate the required Block fields: `content`, `parentHashes`, `contentHash`, and `blockHash`
- enforce the required hash encoding for `contentHash`, `blockHash`, and entries in `parentHashes`
- normalize `parentHashes` by lowercasing valid hashes, discarding invalid entries, removing duplicates, and sorting the retained entries before `blockHash` derivation
- canonicalize `content` according to ALX canonicalization rules
- compute `contentHash` as `keccak256(canonicalize(content))`
- compute `blockHash` as `keccak256(canonicalize({content, parentHashes}))`
- compare recomputed hash values with declared hash values during verification

- reject structurally invalid Blocks
- support root Blocks with empty `parentHashes`
- support derived Blocks with declared parent references
- detect cycles during derivation Graph traversal
- distinguish unresolved parents from declared external parents when verification context is provided
- compute structural Graph metrics such as `pathCount` when Attribution Traces are generated

Canonical test vectors determine conformance by providing shared inputs and expected outputs for canonicalization, hash derivation, Block validation, Graph validation, Attribution Traces, and verification receipts.

Implementations MAY provide additional capabilities such as SDKs, storage adapters, schema validation, attestation verification, indexing, registry publishing, anchoring, governance workflows, visualization tools, or application-specific policy engines. These capabilities remain outside core conformance unless explicitly required by the protocol specification or an adopted extension.

Conformance is structural. A conforming implementation verifies protocol data according to the ALX Protocol rules. Conformance does not establish semantic truth, content quality, authorship, legal validity, policy compliance, storage availability, or institutional trust.

Summary

Specification and Conformance define the shared rules that make ALX Protocol independently implementable and interoperable across conforming systems. Canonicalization, hash derivation, Block structure, parent handling, Graph traversal, Attribution Traces, verification receipts, and extension compatibility provide a common foundation for deterministic identity and structural verification.

The protocol surface remains intentionally minimal. A conforming implementation creates, verifies, and traverses Blocks using the shared rules defined by the canonical specification, schemas, and test vectors. Equivalent protocol inputs produce equivalent identities and verification results across independent implementations.

Conformance enables portable verification. Blocks created in one conforming environment can be verified, referenced, and composed in another without requiring a shared storage system, hosted service, blockchain, registry, or application framework.

The specification defines structural behavior only. ALX Protocol standardizes deterministic identity, declared lineage, derivation Graph relationships, structural attribution, and verification outputs. Applications and external systems remain responsible for storage, semantic interpretation, trust, governance, policy enforcement, legal meaning, and operational use.

Related Documents

The ALX Protocol specification, schemas, canonical test vectors, extension manifests, and related technical documents are available at **https://alx.pro**

Conclusion

ALX Protocol is an open protocol for composable digital artifacts. It provides a shared structural foundation for representing, identifying, composing, and verifying digital artifacts across applications, organizations, and execution environments.

The Block is the protocol primitive. Deterministic content-only identity, lineage-bound Block identity, and declared parent relationships allow conforming implementations to verify and traverse derivation Graphs through shared protocol rules.

The protocol intentionally separates structure from interpretation. ALX defines deterministic identity, declared lineage, and structural verification, while applications and external systems determine semantic meaning, trust, governance, policy, legal interpretation, storage, and operational behavior. This separation enables independent implementations to interoperate through a common protocol while remaining free to innovate above the Block layer.