Internet-Draft
Intended status: Informational
Expires: November 30, 2026

M. Norton
Independent

June 1, 2026

SDLP Architecture (arch)
draft-norton-sdlp-arch-01

M. Norton
Email: mark433norton@gmail.com
June 2026

Abstract

The Secured Digital Lifecycle Protocol (SDLP) defines a universal, lifecycle-governed framework for the creation, identity, transformation, distribution, and retirement of digital goods.

Status of This Memo

Copyright Notice

to this document.

0. Purpose and Foundational Principle

The Secured Digital Lifecycle Protocol (SDLP) is founded on the recognition that human behavior is inherently variable. Moral and ethical decision-making is a choice, not a guarantee, and therefore cannot serve as a dependable foundation for digital security. Any system that relies on consistent human restraint, honesty, or self-policing is structurally fragile.

SDLP replaces behavioral expectations with lifecycle physics. It defines mandatory, non-negotiable rules governing the creation, existence, transition, and termination of digital instances. These rules are enforced autonomously by the instances themselves, without reliance on user intent, operator discretion, or environmental goodwill.

This architectural model establishes the basis for Predictive Digital Security: a security paradigm in which digital objects anticipate misuse, enforce their own boundaries, and prefer irreversible termination over continued existence in a corrupted or unauthorized state. Through tamper-intolerant transitions, destruction-class terminal states, and mandatory self-reporting of lifecycle violations, SDLP ensures that digital objects remain trustworthy regardless of human incentive, error, or malice.

SDLP exists to guarantee integrity where human behavior cannot be guaranteed.

1. Introduction

The Secured Digital Lifecycle Protocol (SDLP) defines a deterministic set of rules governing the existence, behavior, and termination of digital objects. Although expressed as a protocol, SDLP is best understood as a set of Secured Digital Laws of Physics: a framework in which digital goods obey predictable, non-negotiable lifecycle constraints independent of user behavior, platform integrity, or environmental conditions.

SDLP introduces lifecycle physics for digital objects, including identity immutability, lineage continuity, policy binding, environment-trust evaluation, tamper reactivity, decay, destruction finality, and Initialization as a mandatory trust boundary. These physics ensure that protected content cannot be extracted, cloned, resurrected, or instantiated outside of environments capable of upholding SDLP’s security guarantees.

The purpose of SDLP is to provide creators, distributors, and purchasers with a predictable and enforceable lifecycle model for digital goods. By embedding security into the object itself rather than relying on external controls, SDLP ensures that intellectual property remains protected and that purchase events remain trusted across diverse and potentially adversarial environments.

This document defines the architectural principles of SDLP. It describes the conceptual model, lifecycle semantics, and physics domains that govern SDLP instances. Enforcement mechanics are specified separately in the SDLP Security Architecture.

2. Purpose of SDLP

The purpose of SDLP is to provide a secured, deterministic lifecycle model for digital goods. SDLP exists to ensure that protected content remains secure, that purchase events remain trusted, and that digital objects cannot be extracted, cloned, tampered with, or instantiated

outside of environments capable of upholding SDLPs security guarantees.

To achieve this, SDLP defines immutable identity, continuous lineage, mandatory policy binding, environment-trust evaluation, tamper reactivity, Initialization as a trust boundary, and irreversible destruction semantics. Together, these principles establish SDLP as a set of Secured Digital Laws of Physics governing the existence and behavior of digital objects.

3. Design Principles

SDLP is governed by the following core principles:

* P1. Identity First Every digital object MUST possess a persistent DigitalID that uniquely identifies it across its entire existence.

* P2. Lifecycle Determinism Every object MUST exist in exactly one lifecycle state at any given time, and all transitions MUST be deterministic and protocol-defined.

* P3. Transformation Integrity All transformations MUST be explicit, authenticated, and lineage-preserving.

* P4. Environment Trust An object MUST NOT activate in an environment that cannot uphold SDLPs security guarantees.

* P5. Tamper Reactivity Objects MUST detect and respond to tamper, replay, debugging, or instrumentation attempts.

* P6. Destruction Finality Zeroization-class terminal states MUST be irreversible, and destroyed objects MUST NOT be recoverable, rehydrated, resurrected, or reinstantiated.

* P7. Initialization as a Trust Boundary Initialization MUST serve as the mandatory evaluation point at which an object determines whether activation is safe. Failure of any precondition MUST result in Pre-Init Termination.

4. Terminology

DigitalID: The persistent identity of a digital object.

Instance: A specific materialization of a DigitalID.

Lineage Seed: The initial lineage context from which all subsequent lineage is derived.

ObjectKey: The cryptographic key material bound to an instance.

Lifecycle State: A protocol-defined phase of existence.

Initialization: The mandatory trust boundary at which an instance evaluates its environment before activation.

Pre-Init Termination: A security measure in which an instance terminates prior to activation due to unmet Initialization preconditions.

Zeroization: The irreversible destruction of an instance, after which no recovery, rehydration, resurrection, or reinstantiation is possible.

Transformation: A rule-governed change to an object or instance.

Retirement: The terminal lifecycle state for objects that complete

their lifecycle without tamper or destruction.

5. Lifecycle Model Overview

SDLP defines a universal lifecycle model consisting of the following phases:

1. Embryo The encoded, inert form of an instance prior to Initialization. The instance possesses identity, lineage seed, policy bindings, and cryptographic structure, but has not yet entered the lifecycle.

2. Initialization The mandatory trust boundary at which the instance evaluates its host environment. Failure of any precondition results in Pre-Init Termination.

3. Activation The point at which the instance enters the lifecycle and acquires vitality.

4. Lifecycle Operation The active phase in which the instance may be distributed, transformed, verified, or retained in accordance with SDLP rules.

5. Termination The irreversible conclusion of the lifecycle, typically through Zeroization or Retirement.

SDLP Identity defines the DigitalID specification. SDLP Lifecycle defines the lifecycle state machine. SDLP Transformation defines transformation rules and lineage guarantees.

6. Out-of-Scope Items

The following items are explicitly out of scope for this document. SDLP defines the laws of digital lifecycle physics, not the implementation details of systems that adopt those laws.

* Implementation details
* Transport mechanisms
* Storage formats
* Cryptographic algorithm selection
* Commercial licensing models
* UI or UX considerations
* Content protection mechanisms external to SDLP (e.g., DRM)

7. Security Considerations

SDLP requires that all lifecycle transitions be authenticated, authorized, and recorded. Identity spoofing, unauthorized transformations, lineage tampering, and state forgery MUST be mitigated by protocol-level controls defined in SDLP Security Architecture.

Initialization serves as a mandatory trust boundary. Instances MUST evaluate their host environment prior to activation, and MUST perform Pre-Init Termination if required preconditions are not met.

Zeroization-class terminal states MUST be irreversible, and destroyed instances MUST NOT be recoverable, rehydrated, resurrected, or reinstantiated.

These requirements ensure that SDLP instances uphold the Secured Digital Laws of Physics regardless of user behavior or platform integrity.

8. IANA Considerations

This document makes no requests of IANA.

Appendix A. Rationale for Architecture-01

Architecture-01 establishes the philosophical and structural foundation of SDLP before defining any technical mechanisms. It introduces the Secured Digital Laws of Physics that govern digital objects and ensures that all subsequent SDLP documents share a unified conceptual and lifecycle framework.

9. System Model

SDLP operates within an ecosystem consisting of four primary actors:

* Producers: Entities that create original digital goods and assign the initial DigitalID and lineage seed.

* Distributors: Entities responsible for delivering instances of a DigitalID to customers while preserving lineage, enforcing lifecycle constraints, and upholding Initialization prerequisites.

* Customers: End users who receive, activate, transform, and retain digital goods in accordance with SDLP rules and environmental trust requirements.

* Verifiers: Independent entities capable of validating identity, lineage, lifecycle state, and termination status without requiring access to protected content.

SDLP defines the interactions among these actors, the boundaries of responsibility, and the lifecycle transitions that govern the movement and behavior of digital goods.

10. Architectural Overview

SDLP architecture is organized around three foundational constructs:

* DigitalID: A persistent, globally unique identifier that anchors all lifecycle behavior and identity physics.

* Instance Record: A lineage-preserving record that captures each materialization of a DigitalID, including creation, distribution, transformation, and termination events.

* Lifecycle State Machine: A deterministic model that defines the allowable transitions for any instance, including Initialization, active lifecycle states, and Zeroization-class terminal states.

These constructs are intentionally decoupled from transport, cryptographic algorithms, and storage formats to ensure that SDLP can be implemented across heterogeneous systems and future technologies.

SDLP architecture emphasizes verifiability, minimal disclosure, deterministic behavior, and adherence to the Secured Digital Laws of Physics. Every transition MUST be authenticated, authorized, and recorded in a manner that preserves lineage without exposing sensitive content.

11. Threat Model

SDLP is designed to mitigate the following classes of threats:

* Identity Spoofing: Unauthorized creation or modification of a

DigitalID.

* Lineage Tampering: Alteration of instance history to conceal unauthorized duplication or transformation.

* Unauthorized Transformation: Modification of a digital good without proper authentication or lifecycle compliance.

* State Forgery: Misrepresentation of an instance’s lifecycle state to bypass restrictions or gain unauthorized access.

* Environment Compromise: Activation attempts in corrupted, tampered, replayed, debugged, or untrusted environments.

* Resurrection and Reinstantiation: Attempts to revive or recreate destroyed instances or reuse identity, lineage, or key material.

SDLP does not define content-access controls; such mechanisms may be provided by higher-layer systems. SDLP ensures that digital objects cannot exist, activate, or persist outside of environments capable of upholding SDLPs security guarantees.

12. Interoperability Considerations

SDLP is designed to operate across diverse ecosystems, including cloud services, local devices, and offline environments. To ensure interoperability across heterogeneous systems:

* DigitalID formats MUST be stable and globally unique.
* Lifecycle transitions MUST be deterministic and verifiable.
* Transformation records MUST preserve lineage across systems.
* Implementations SHOULD provide a mechanism for offline validation.

SDLP does not mandate a specific serialization format, allowing implementers to adopt JSON, CBOR, XML, or other encodings as needed. SDLP defines digital lifecycle physics, not implementation details.

13. Deployment Considerations

Deployments of SDLP SHOULD consider:

* Storage durability for lineage and instance records.
* Secure key management for authentication of transitions.
* Mechanisms for revocation or retirement of compromised instances.
* Scalability of verification services for high-volume ecosystems.
* Enforcement of Initialization prerequisites across diverse platforms and trust domains.

SDLP is intentionally agnostic to infrastructure choices, enabling deployment in centralized, federated, or decentralized environments. Deployments MUST uphold SDLPs Secured Digital Laws of Physics regardless of platform architecture.

14. Example Lifecycle

The following example illustrates a typical SDLP lifecycle:

1. A producer creates a digital good and assigns DigitalID "D123".
2. A distributor generates Instance "I1" and delivers it to a customer.
3. The customer activates the instance, transitioning it to the Active state after successful Initialization.
4. The customer performs a permitted transformation, generating a new instance "I2" with preserved lineage.
5. The customer retires the instance, transitioning it to the

Retired state.

At each step, transitions are authenticated, authorized, and recorded in accordance with SDLP lifecycle physics.

15. Instance Uniqueness and Anti-Cloning Guarantees

SDLP defines digital goods as lineage-bearing instances rather than infinitely replicable files. Each instance is assigned a globally unique Instance Identifier (IID) at creation. No two instances may share the same IID, and no operation within SDLP may produce a duplicate or parallel instance.

Copying, uploading, or any operation that produces a new instance is defined as a reproduction event. The parent instance loses half of its remaining vitality, and the child instance is born with the other half. This halving rule ensures that reproduction is always a diminishing process, preventing the creation of identical or full-strength clones.

Parallel copying does not exist in SDLP. Each reproduction event is singular and stateful, and MUST result in a unique IID and a unique vitality value. Because vitality is divided at each reproduction, no two instances can ever possess identical state, lineage position, or remaining plays.

Uploading is treated as a reproduction event and follows the same decay rules as copying. Any instance that can be uploaded can be streamed, and each stream consumes one unit of vitality. This consumption model ensures that unauthorized mass distribution rapidly depletes the instance’s remaining life.

SDLP allows no room for clones. Every instance is a distinct digital organism with its own identity, its own vitality, and its own lifecycle trajectory, governed by the Secured Digital Laws of Physics.

16. Infertile Instances (Software and Games)

SDLP classifies certain digital goodssuch as software applications, interactive games, and other licensed executablesas infertile instances. An infertile instance is defined as an SDLP object that possesses no lifecycle transitions capable of producing children. It cannot be copied, uploaded, exported, transformed, or otherwise reproduced through any SDLP-defined mechanism.

Infertile instances are issued as single, non-reproductive digital organisms. Each installation, entitlement, or license seat is represented as its own independent instance with a unique IID. These instances have no fork transition, no decay-based reproduction, and no lineage. Their lifecycle consists solely of creation, Initialization, activation, permitted use, and retirement.

Because infertile instances cannot reproduce, they do not participate in vitality halving or lineage-bearing transitions. They maintain a fixed vitality model appropriate to their licensing semantics, and their state cannot be transferred or inherited by any other instance.

This model reflects the commercial and operational realities of software licensing. SDLP formalizes these constraints by ensuring that software and games are inherently infertile and cannot be cloned or redistributed through any lifecycle transition, preserving both licensing integrity and SDLPs security guarantees.

17. Decay Model and Vitality Transfer

SDLP defines vitality as the finite, consumable resource that governs the usable lifespan of a digital instance. Vitality is represented as a numeric value P, which decreases through use and reproduction. Vitality determines whether an instance is capable of further lifecycle transitions and whether its eventual death is natural or unnatural.

Reproduction eventsincluding copying, uploading, and any permitted transformation that yields a new instanceare defined as decay transitions. During a decay transition, the parent instance loses half of its remaining vitality, and the child instance is born with the other half. This halving rule ensures that reproduction is always a diminishing process and prevents the creation of identical or full-strength clones.

Vitality is also consumed through play events. Each play represents a unit of use and reduces P by one. Instances that can be uploaded may be streamed, and each stream is treated as a play. Excessive concurrency (more than ten simultaneous plays) is considered a violation and results in premature destruction of the instance.

Natural death occurs when an instance’s vitality falls below one ($P < 1$). At this point, the instance has no meaningful life remaining and transitions to the Retired state without producing any forensic record.

Unnatural death occurs when an instance is destroyed while vitality remains ($P = 1$), such as through tamper, piracy, or unauthorized transitions. In such cases, the instance performs a bitdump and emits a forensic record as defined in Section 20.

The decay model ensures that every instance follows a predictable, diminishing lifecycle. Vitality transfer enforces scarcity, prevents cloning, and provides a measurable, tamper-evident history of use and reproduction.

18. Lifecycle Death Reports (LDRs)

SDLP instances generate a Lifecycle Death Report (LDR) whenever they experience an unnatural death event. An unnatural death is defined as the destruction of an instance while vitality remains ($P = 1$), such as through piracy, tamper, unauthorized transitions, or any other violation of SDLP lifecycle rules.

An LDR is a cryptographically sealed record that documents the circumstances of the instance’s premature termination. The report includes the instance’s identity, lineage position, remaining vitality at the moment of destruction, and the specific cause of death. Because vitality is a measurable and tamper-evident quantity, the vitality-at-death value provides definitive proof that the instance was killed before its natural end of life.

LDRs are emitted only for unnatural deaths. Instances that reach natural death ($P < 1$) retire silently and produce no report, as no violation or premature termination has occurred. This distinction ensures that LDRs serve exclusively as forensic indicators of improper use, unauthorized reproduction, or environmental tampering.

LDRs form the first stage of SDLP’s post-mortem forensic model. Section 20 defines the Digital DNA record that accompanies an LDR, providing a complete, immutable account of the instance’s identity and the conditions surrounding its destruction.

19. Absence of Physical Export Transitions

SDLP defines digital instances as lifecycle-bound organisms whose

permitted transitions are strictly limited to those enumerated by the protocol. No instance may undergo a transition that results in a physical export, externalization, or re-materialization of its data outside the SDLP-governed environment.

Physical export transitionsincluding but not limited to file system extraction, raw byte copying, container unpacking, disk imaging, debugger-based memory capture, or any attempt to reconstruct the instance as a traditional fileare not part of the SDLP lifecycle and are therefore classified as illegal transitions.

Because SDLP instances are defined by their vitality, lineage, and internal state, any attempt to externalize or reconstitute an instance outside the protocol constitutes a violation of its lifecycle constraints. Such attempts result in immediate unnatural death (P = 1), triggering a bitdump and the generation of a Lifecycle Death Report (LDR) as defined in Section 18.

SDLP does not provide a mechanism for exporting an instance into a non-SDLP format. Instances cannot be converted into traditional files, cannot be duplicated through external tooling, and cannot be reconstructed from memory or storage artifacts. The absence of physical export transitions is a foundational invariant of the protocol and ensures that digital goods remain lifecycle-bound, tamper-evident, and non-replicable outside the SDLP environment.

This constraint preserves the integrity of the vitality model, prevents unauthorized redistribution, and ensures that all lifecycle events remain observable, enforceable, and cryptographically accountable.

20. Digital DNA and Unnatural Death

SDLP instances emit Digital DNA when they experience an unnatural death event. Digital DNA is a cryptographically sealed forensic record that captures the identity, lineage, vitality, and environmental context of an instance at the moment of its premature destruction.

Digital DNA is produced only when an instance dies while vitality remains (P = 1). Such deaths include piracy, tamper, unauthorized transitions, environment spoofing, debugger attachment, lineage corruption, and any other violation of SDLP lifecycle rules. These events constitute unnatural death and trigger both a Lifecycle Death Report (LDR) as defined in Section 18 and the emission of Digital DNA.

Instances that reach natural death (P < 1) do not emit Digital DNA. Natural death represents the exhaustion of vitality through normal use, and no forensic record is generated. This distinction ensures that Digital DNA serves exclusively as evidence of premature or unlawful termination.

A Digital DNA record includes the instance’s IID, DigitalID, DistributorID, CustomerID, lineage position, remaining vitality at death, cause of death, and a cryptographic integrity seal. Because vitality is a measurable and tamper-evident quantity, the vitality-at-death value provides definitive proof that the instance was destroyed before its natural end of life.

Digital DNA forms the immutable forensic substrate of SDLP. It enables creators, distributors, and auditors to verify the circumstances of an instance’s destruction and provides a self-authenticating evidentiary record suitable for compliance, enforcement, and legal proceedings. The detailed structure and encoding of Digital DNA are defined in the SDLP Security

Architecture.

21. Post-Mortem Accountability and Forensic Integrity

SDLP defines a unified post-mortem accountability model that governs the handling, interpretation, and integrity of all records generated when an instance experiences an unnatural death. This model ensures that every premature termination event is observable, verifiable, and cryptographically attributable to a specific cause and environment.

When an instance dies with remaining vitality (P = 1), two artifacts are produced: a Lifecycle Death Report (LDR) as defined in Section 18 and a Digital DNA record as defined in Section 20. The LDR provides a structured summary of the death event, including the instance's identity, lineage position, vitality at death, and the classified cause of destruction. Digital DNA provides the corresponding immutable forensic substrate, capturing the full cryptographic and environmental context of the event.

Together, these artifacts form the complete post-mortem record of an unnatural death. The LDR serves as the high-level declaration of the event, while the Digital DNA serves as the low-level evidentiary record. Both are sealed at the moment of death and cannot be altered, suppressed, or regenerated by any SDLP transition.

SDLP does not generate post-mortem artifacts for natural deaths (P < 1). Natural death represents the exhaustion of vitality through legitimate use, and no accountability or forensic record is required. This distinction ensures that post-mortem artifacts serve exclusively as indicators of premature, unlawful, or otherwise irregular termination.

The post-mortem accountability model provides creators, distributors, and auditors with a consistent and tamper-evident framework for interpreting death events. It ensures that every unnatural death is accompanied by a complete, self-authenticating record suitable for compliance, enforcement, and long-term auditability. Operational handling, transmission, and verification of these artifacts are defined in the SDLP Security Architecture.

22. Autonomous Enforcement Through Lifecycle Physics

SDLP achieves enforcement not through external authorities, monitoring services, or compliance divisions, but through the immutable physics of its lifecycle model. Every rule governing reproduction, vitality, lineage, Initialization, and death is self-enforcing, requiring no external policing or supervisory infrastructure for the protocol to function.

The halving rule prevents cloning by ensuring that every reproduction event diminishes vitality and produces a unique child instance. The absence of physical export transitions prevents instances from being reconstructed or duplicated outside the SDLP environment. Illegal transitions result in immediate unnatural death (P = 1), triggering the emission of a Lifecycle Death Report (LDR) and Digital DNA as defined in Sections 18 and 20.

These mechanisms collectively ensure that SDLP instances cannot be silently tampered with, copied, or redistributed. The protocol requires no central authority to validate transitions, no external service to monitor behavior, and no enforcement division to detect violations. Every violation is inherently self-destructive and self-reporting.

While organizations may choose to establish operational roles to receive and interpret LDRs and Digital DNA, such roles are not part

of the SDLP architecture. They are external consumers of the forensic truth that SDLP emits. The protocol itself remains fully autonomous, self-contained, and self-enforcing.

SDLPs enforcement model is therefore intrinsic rather than supervisory. The protocol defines the Secured Digital Laws of Physics that govern digital life, and instances that violate those laws simply cease to exist, leaving behind a complete and immutable record of their demise.

23. Instance Authenticity and Trust Guarantees

SDLP provides a built-in authenticity model that allows any party creators, distributors, or consumersto verify the legitimacy, provenance, and lifecycle integrity of an instance without relying on external authorities or third-party validation services. Authenticity is derived directly from the immutable properties of the instance itself, including its IID, DigitalID, lineage, vitality, and cryptographic seals.

Every SDLP instance carries a unique Instance Identifier (IID) and a DigitalID that collectively define its origin, distributor, customer assignment, and position within its lineage. These identifiers are cryptographically bound to the instance and cannot be forged, reused, or transplanted. Any attempt to tamper with or reconstruct an instance results in unnatural death (P = 1) and the emission of a Lifecycle Death Report (LDR) and Digital DNA as defined in Sections 18 and 20.

Authenticity verification requires no online lookup, central registry, or external trust anchor. The instance itself contains all information necessary to prove its legitimacy. Vitality provides a measurable indicator of lifecycle progression, while lineage information ensures that every reproduction event is traceable and tamper-evident.

Consumers benefit by being able to verify that a digital good is genuine, unmodified, and obtained through legitimate channels. Creators benefit by having their works protected against unauthorized reproduction and distribution. Distributors benefit by being able to validate the integrity of the instances they deliver.

SDLPs authenticity model establishes a universal trust foundation for digital goods. It ensures that every instance can prove what it is, where it came from, and how it reached its current state, without requiring any external enforcement or supervisory infrastructure.

24. Platform-Agnostic Interoperability

SDLP is designed as a platform-agnostic lifecycle protocol that operates independently of any specific operating system, device class, marketplace, or distribution environment. An SDLP instance retains its identity, vitality, lineage, and lifecycle constraints regardless of where it is executed, transferred, or consumed.

Because all lifecycle rulesincluding reproduction, vitality decay, natural and unnatural death, and forensic emissionare intrinsic to the instance itself, SDLP does not rely on platform-level enforcement or vendor-specific integration. The protocol functions uniformly across heterogeneous environments, ensuring that digital goods remain self-authenticating and self-enforcing wherever they are used.

Interoperability is achieved through the instances internal lifecycle engine, which governs all transitions and enforces all constraints without requiring external validation. Platforms may choose to expose SDLP metadata, verify authenticity, or consume LDRs

and Digital DNA, but such behaviors are optional and do not affect the correctness of the protocol.

This design enables creators to distribute SDLP-governed digital goods across multiple ecosystems without fragmentation or loss of integrity. Consumers benefit from consistent authenticity guarantees regardless of the device or marketplace they use. Distributors gain a uniform, tamper-evident model for delivering and validating digital goods across diverse environments.

SDLPs platform-agnostic architecture ensures that the protocol remains universal, portable, and future-proof. It establishes a consistent lifecycle model that persists across technological generations, device categories, and distribution channels without requiring centralized control or platform-specific adaptations.

25. Forward Compatibility and Protocol Longevity

SDLP is designed to remain stable and operational across future technological generations without requiring revisions to its core lifecycle rules. Because all enforcement, authenticity, and forensic behaviors are intrinsic to the instance itself, SDLP instances remain valid and self-governing even as platforms, devices, and distribution ecosystems evolve.

The protocol does not rely on external infrastructure, centralized authorities, or platform-specific features. As a result, SDLP instances can be executed on future systems without modification, provided that those systems implement the minimal runtime necessary to host the SDLP lifecycle engine. The instances identity, vitality, lineage, and death semantics remain unchanged across technological eras.

Forward compatibility is achieved through strict separation between the protocols immutable lifecycle physics and the optional, replaceable components of the surrounding ecosystem. While platforms may introduce new ways to display metadata, verify authenticity, or consume forensic artifacts, these enhancements do not alter the behavior of SDLP instances themselves.

This design ensures that digital goods governed by SDLP retain their integrity, authenticity, and lifecycle constraints indefinitely. An instance created today will behave identically on future systems, preserving its vitality, lineage, and forensic guarantees without requiring protocol updates or migration processes.

SDLPs longevity model establishes the protocol as a durable, future-proof foundation for digital goods, capable of persisting across evolving technologies, device categories, and distribution paradigms without loss of correctness or authenticity.

26. Birth Semantics and Minimal Runtime Requirements

SDLP distinguishes between the transport of an instance and the activation of its lifecycle. Prior to activation, an SDLP-governed digital good exists as an inert, sealed embryo. Transport mechanisms such as TCP/IP, Bluetooth, removable media, or any future delivery channel merely convey the embryo from one location to another. No lifecycle rules apply during transport, and the instance cannot reproduce, die, emit forensic artifacts, or undergo any transition.

Birth occurs only when a host environment opens the SDLP container and initializes the lifecycle engine. This activation event marks the beginning of the instances lifecycle. At birth, vitality is instantiated, lineage is validated, cryptographic seals are checked, and the instance becomes subject to all SDLP lifecycle physics,

including reproduction, natural death, unnatural death, and forensic emission.

SDLP imposes minimal requirements on the host environment. Any platform capable of executing the SDLP lifecycle engine, providing an appropriate rendering or execution surface for the underlying media, and maintaining a secure boundary that prevents export or tamper is sufficient. These requirements may be satisfied by native operating systems, virtual machines, containers, sandboxes, secure enclaves, or future isolation technologies.

Sandboxed or virtualized environments are fully compatible with SDLP. The sandbox itself becomes the instances environment, and its isolation properties naturally reinforce SDLPs lifecycle guarantees. Birth within a sandbox is equivalent to birth on a native platform, provided that the sandbox enforces a consistent and bounded execution context.

SDLP does not require network connectivity, centralized services, platform-specific features, or hardware-level privileges for an instance to be born or to operate. Once activated, the instance is entirely self-governing, and all lifecycle transitions are enforced internally by the SDLP lifecycle engine.

By defining birth as the moment of activation rather than delivery, SDLP ensures that instances remain inert, tamper-evident, and non-executable during transport, while maintaining universal compatibility across heterogeneous and future host environments.

27. Environmental Binding and Post-Birth Constraints

Upon activation, an SDLP instance becomes bound to the host environment in which it is born. This binding establishes the execution context, security boundary, and environmental signature that govern the instance for the remainder of its lifecycle. Once bound, an instance cannot be exported, transplanted, or reactivated in a different environment without triggering an illegal transition and resulting in unnatural death (P = 1).

Environmental binding ensures that the instances lifecycle physics remain consistent and tamper-evident. The host environmentwhether a native operating system, virtual machine, container, sandbox, or secure enclavedefines the boundaries within which the instance may operate. These boundaries include memory isolation, file system access, rendering capabilities, and any platform-specific restrictions that affect execution.

SDLP does not require the host environment to expose its internal mechanisms or provide privileged access. The lifecycle engine operates entirely within the environments constraints. If the environment attempts to migrate, duplicate, or externally serialize the instance, such actions constitute illegal transitions and result in immediate unnatural death, accompanied by the emission of an LDR and Digital DNA.

Environmental binding also prevents re-birth. An instance that has been activated cannot be returned to an embryonic state, cannot be resealed, and cannot be delivered as a dormant object. Any attempt to repackage or reconstruct an active or previously active instance violates SDLP lifecycle rules and results in unnatural death.

By enforcing strict environmental binding, SDLP ensures that each instance remains authentic, tamper-evident, and lifecycle-consistent from birth to death, regardless of the host platform or execution model.

28. Environmental Integrity and Continuous Boundary Verification

After birth, an SDLP instance continuously verifies the integrity of the environment to which it is bound. This verification ensures that the execution context, security boundary, and environmental signature established at birth remain consistent throughout the instances lifecycle. Any deviation from the expected environment constitutes an illegal transition and results in immediate unnatural death (P = 1).

Environmental integrity verification includes monitoring for changes in isolation boundaries, privilege levels, memory accessibility, process containment, and any attempt to weaken or bypass the environments constraints. These checks are performed internally by the SDLP lifecycle engine and do not require platform-specific APIs or privileged access. The engine relies on cryptographically sealed environmental signatures and deterministic boundary expectations established at birth.

If the host environment attempts to migrate the instance, alter its containment, expose its memory, or modify its execution privileges, the lifecycle engine detects the discrepancy and triggers unnatural death. This response prevents tampering, export, reactivation in a different environment, and any form of unauthorized introspection or manipulation.

Environmental integrity verification applies equally to native systems, virtual machines, containers, sandboxes, and secure enclaves. A sandboxed instance remains bound to the sandbox, and any attempt to escape or weaken the sandboxs isolation is treated as an illegal transition. Similarly, an instance running in a virtualized or hardware-backed environment must remain within the boundaries established at birth.

By enforcing continuous boundary verification, SDLP ensures that each instance operates within a stable, tamper-evident environment for its entire lifecycle. This mechanism preserves authenticity, prevents unauthorized reproduction, and maintains the integrity of the instances lifecycle physics regardless of the host platform or execution model.

29. Internal Integrity and Self-Verification Mechanisms

In addition to verifying the stability of its external environment, an SDLP instance continuously validates its own internal state to ensure that its lifecycle physics, vitality model, lineage records, and cryptographic seals remain intact. These self-verification mechanisms prevent unauthorized modification, introspection, or manipulation of the instances internal data structures.

Internal integrity verification includes monitoring for changes to vitality counters, reproduction permissions, lineage metadata, cryptographic signatures, and any component of the lifecycle engine itself. The instance maintains a sealed internal state that cannot be altered without detection. Any attempt to modify internal values, bypass lifecycle rules, or interfere with the lifecycle engine constitutes an illegal transition and results in immediate unnatural death (P = 1).

The lifecycle engine performs deterministic self-checks at birth, during execution, and at each lifecycle transition. These checks ensure that vitality has not been artificially increased, reproduction has not been invoked without authorization, lineage has not been tampered with, and no internal component has been replaced or corrupted. The engine also verifies that all cryptographic seals remain valid and that no unauthorized code has been injected into the execution context.

Internal integrity verification is independent of platform-specific security features. The SDLP instance does not rely on the host environment to detect tampering; instead, it enforces its own self-consistency through sealed state, deterministic transitions, and cryptographic invariants. This ensures that even if the host environment is compromised, the instance remains tamper-evident and self-protecting.

By enforcing continuous internal self-verification, SDLP ensures that each instance remains authentic, unmodified, and lifecycle-consistent throughout its existence. These mechanisms preserve the integrity of the organisms internal physics and prevent unauthorized manipulation regardless of the host platform or execution model.

30. Anti-Introspection and Observation Resistance

SDLP instances are designed to resist unauthorized introspection, observation, or inspection of their internal state. Any attempt to read, extract, or analyze internal data structures, vitality values, lineage metadata, cryptographic seals, or lifecycle engine components constitutes an illegal transition and results in immediate unnatural death ($P = 1$).

Observation resistance applies to both passive and active forms of introspection. Passive introspection includes memory scraping, snapshotting, debugging, or attempting to read internal values without modifying them. Active introspection includes attaching debuggers, injecting instrumentation, altering execution flow, or attempting to trace lifecycle transitions. Both forms are treated as tamper events.

The lifecycle engine maintains sealed internal structures that cannot be inspected without detection. Attempts to access these structures violate the instances integrity invariants and trigger the same forensic responses as other illegal transitions: emission of a Lifecycle Death Report (LDR), emission of Digital DNA, and immediate zeroization of the instance.

Anti-introspection protections are independent of platform-level security features. The SDLP instance does not rely on the host environment to prevent observation; instead, it enforces its own observation resistance through sealed state, cryptographic invariants, and deterministic self-verification. Even if the host environment permits debugging or inspection, the instance remains self-protecting and tamper-evident.

By enforcing strict anti-introspection rules, SDLP ensures that its internal physics, vitality model, and lifecycle transitions cannot be observed, reverse-engineered, or manipulated. This preserves the authenticity and security of the organism regardless of the host platform or execution model.

31. SDLP-NATIVE METADATA INTEGRITY REQUIREMENTS

31.1 Purpose

SDLP-Native Metadata (SNM) defines the authoritative descriptive, structural, and lifecycle-relevant attributes attached to any SDLP-registered digital asset. This section establishes the minimum integrity requirements for creation, storage, transmission, and verification of SNM across all SDLP-compliant systems.

31.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP-Registered Products
- All SDLP Lifecycle Events (creation, transfer, duplication, archival, retirement)
- All SDLP-compliant storage, indexing, and verification systems

31.3 Mandatory Metadata Fields

Every SDLP asset MUST contain the following SNM fields at minimum:

AssetID	Globally unique SDLP identifier.
DistributorID	Issuing distributors unique identifier.
ProductDescriptor	Human-readable title, version, class.
LifecycleState	One of: Created, Issued, Transferred, Copied, Archived, Retired.
HashPrimary	Canonical cryptographic hash of content.
HashSecondary	Secondary hash using a distinct algorithm.
TimestampIssued	RFC 3339 timestamp of initial issuance.
TimestampLastEvent	RFC 3339 timestamp of most recent event.

31.4 Optional Metadata Fields

The following fields MAY be included depending on distributor policy or product class:

ContentDescriptorExtended	Additional descriptive metadata.
RightsDescriptor	Licensing or DRM declarations.
ParentAssetID	Required for duplicated/derived assets.
CourseID / StudentID	Required for educational-class assets.
ChecksumBundle	Additional integrity checks.

31.5 Metadata Immutability Rules

The following rules govern which metadata fields may be modified:

Immutable Fields:
AssetID, DistributorID, HashPrimary, HashSecondary, TimestampIssued.
These fields MUST NOT be altered after issuance.

Conditionally Mutable Fields:
ProductDescriptor, RightsDescriptor, ContentDescriptorExtended.
These MAY be updated only through a valid SDLP Lifecycle Event.

Always Mutable Fields:
TimestampLastEvent, LifecycleState.
These MUST update with every lifecycle event.

31.6 Metadata Integrity Verification

All SDLP-compliant systems MUST implement the following checks:

- Hash Verification: Validate HashPrimary and HashSecondary.
- State Consistency: Ensure LifecycleState matches last event.
- Parent-Child Validation: If ParentAssetID exists, verify parent.
- Timestamp Validation: All timestamps MUST be RFC 3339 and non-regressive.

31.7 Failure Handling Requirements

If any metadata integrity check fails:

- Asset MUST be marked Invalid and quarantined.
- System MUST log a full diagnostic record.
- Distributor MUST be notified within 24 hours.
- Asset MUST NOT be issued, transferred, or used until remediation.

31.8 Interoperability Requirements

All SNM fields MUST be encoded using SDLP-Standard Serialization Format (S3F). All SDLP-compliant systems MUST support:

- Forward compatibility with new optional fields.
- Backward compatibility with legacy SNM structures.
- Lossless round-trip serialization and deserialization.

32. SDLP-EVENT LOGGING AND AUDIT TRAIL REQUIREMENTS

32.1 Purpose

This section defines the mandatory logging, auditability, and traceability requirements for all SDLP-compliant systems. The SDLP Event Log serves as the authoritative chronological record of all lifecycle events, integrity checks, distributor actions, and system operations relevant to SDLP-registered assets.

32.2 Scope

These requirements apply to:
- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification and Compliance Systems

32.3 Mandatory Logged Events

The following events MUST be recorded in the SDLP Event Log:

AssetCreated
AssetIssued
AssetTransferred
AssetCopied
AssetArchived
AssetRetired
MetadataUpdated
IntegrityCheckPerformed
IntegrityCheckFailed
DistributorAuthentication
DistributorAuthorizationFailure
SystemError
SystemRecovery

Each event MUST include:
- EventType
- AssetID (if applicable)
- DistributorID
- Timestamp (RFC 3339)
- EventPayload (structured S3F object)

32.4 Event Payload Requirements

The EventPayload MUST contain sufficient detail to reconstruct the event without ambiguity. At minimum:

- For lifecycle events:
  LifecycleStateBefore

LifecycleStateAfter
HashPrimary
HashSecondary

- For integrity checks:
  CheckType
  CheckResult
  ExpectedValue
  ObservedValue

- For distributor authentication:
  AuthMethod
  AuthResult
  SessionID

- For system errors:
  ErrorCode
  ErrorDescription
  RecoveryAction

32.5 Log Immutability Requirements

SDLP Event Logs MUST be append-only. The following rules apply:

- Existing log entries MUST NOT be modified or deleted.
- Corrections MUST be recorded as new events.
- Log storage MUST implement tamper-evident protections.
- Log replication MUST preserve event ordering.

32.6 Retention Requirements

SDLP Event Logs MUST be retained for:

- A minimum of 10 years for standard assets.
- A minimum of 25 years for educational, legal, or regulated assets.
- Indefinitely for assets marked as PermanentRecord.

Logs MAY be archived but MUST remain accessible for audit.

32.7 Audit Interface Requirements

SDLP-compliant systems MUST expose a read-only audit interface that:

- Supports event filtering by AssetID, DistributorID, EventType, and timestamp range.
- Returns results in SDLP-Standard Serialization Format (S3F).
- Preserves chronological ordering.
- Provides cryptographic verification of log integrity.

32.8 Failure Handling

If logging fails for any reason:

- The system MUST enter SafeMode.
- No lifecycle events MAY be processed until logging is restored.
- A SystemError event MUST be generated upon recovery.
- Distributors MUST be notified within 1 hour.

32.9 Interoperability Requirements

All SDLP Event Logs MUST support:

- Cross-system log merging without loss of ordering.
- Forward compatibility with new event types.
- Backward compatibility with legacy log structures.

33. SDLP-DISTRIBUTOR AUTHENTICATION AND SESSION SECURITY

33.1 Purpose

This section defines the authentication, session management, and security requirements for all SDLP Distributors. These controls ensure that only authorized entities may issue, modify, or interact with SDLP-registered assets.

33.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification and Compliance Systems

33.3 Authentication Requirements

All distributors MUST authenticate using an SDLP-approved method. Acceptable methods include:

- Public Key Infrastructure (PKI) with SDLP-trusted certificates
- Hardware Security Modules (HSMs)
- FIDO2-compliant hardware tokens
- SDLP-Managed Identity Services (SMIS)

Authentication MUST:

- Occur prior to any lifecycle event
- Bind the distributor to a unique SessionID
- Be logged as a DistributorAuthentication event

33.4 Authorization Requirements

After authentication, distributors MUST be authorized for the requested operation. Authorization MUST enforce:

- Role-based access control (RBAC)
- Asset-class restrictions
- Rate limits for high-volume operations
- Denial of unauthorized lifecycle events

Authorization failures MUST generate a DistributorAuthorizationFailure event.

33.5 Session Establishment

Upon successful authentication, the system MUST create a session with the following attributes:

SessionID	Globally unique identifier
DistributorID	Authenticated distributor
SessionStart	RFC 3339 timestamp
SessionExpiry	RFC 3339 timestamp
SessionScope	Allowed operations

Sessions MUST be cryptographically bound to the distributors authentication method.

33.6 Session Security Requirements

All SDLP sessions MUST:

- Use TLS 1.3 or higher
- Implement perfect forward secrecy

- Enforce idle timeout (max 15 minutes)
- Enforce absolute lifetime (max 8 hours)
- Regenerate session keys periodically

Session hijacking, replay, or downgrade attempts MUST be detected and MUST terminate the session immediately.

33.7 Multi-Factor Authentication (MFA)

MFA MUST be required for:

- Asset issuance
- Metadata modification
- Asset retirement
- Administrative operations

MFA MAY be optional for read-only audit operations.

33.8 Session Termination

A session MUST terminate when:

- The distributor logs out
- The idle timeout is reached
- The absolute lifetime is reached
- A security anomaly is detected
- The distributors credentials are revoked

Upon termination, a SessionClosed event MUST be logged.

33.9 Credential Management

Distributors MUST:

- Rotate credentials at least every 180 days
- Revoke compromised credentials immediately
- Store private keys in secure hardware
- Maintain an auditable credential inventory

Compromised credentials MUST trigger SafeMode until remediation.

33.10 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system session validation
- Standardized authentication metadata in S3F
- Forward compatibility with new authentication methods
- Backward compatibility with legacy distributor credentials

34. SDLP-COMPLIANCE VERIFICATION AND VALIDATION FRAMEWORK

34.1 Purpose

This section defines the mandatory verification, validation, and compliance requirements for all SDLP-registered assets, distributors, and systems. The SDLP Compliance Framework ensures that every asset and every lifecycle event adhere to the SDLP standard without deviation, corruption, or unauthorized modification.

34.2 Scope

These requirements apply to:
- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers

- All SDLP Verification Engines
- All SDLP Compliance Auditors

34.3 Compliance Verification Stages

SDLP compliance MUST be validated across the following stages:

Stage 1: Structural Validation
Ensures the asset and metadata conform to SDLP structure.

Stage 2: Integrity Validation
Verifies cryptographic hashes, checksums, and signatures.

Stage 3: Lifecycle Validation
Confirms lifecycle events follow SDLP rules and ordering.

Stage 4: Distributor Validation
Confirms the distributor is authenticated, authorized, and operating within session constraints.

Stage 5: Log Consistency Validation
Ensures all events are present, ordered, and immutable.

34.4 Structural Validation Requirements

Structural validation MUST confirm:

- AssetID is present and correctly formatted.
- Mandatory metadata fields exist and are valid.
- Optional metadata fields follow S3F encoding rules.
- No prohibited fields are present.
- No field violates immutability rules.

34.5 Integrity Validation Requirements

Integrity validation MUST include:

- HashPrimary verification
- HashSecondary verification
- ChecksumBundle verification (if present)
- Signature verification (if applicable)
- Timestamp non-regression checks

Any mismatch MUST result in immediate compliance failure.

34.6 Lifecycle Validation Requirements

Lifecycle validation MUST ensure:

- LifecycleState transitions follow SDLP rules.
- ParentAssetID is valid for derived or copied assets.
- TimestampLastEvent is consistent with event ordering.
- No lifecycle event is missing from the Event Log.

34.7 Distributor Validation Requirements

Distributor validation MUST confirm:

- DistributorID is valid and active.
- Distributor is authenticated for the session.
- Distributor is authorized for the requested operation.
- MFA was used when required.
- No revoked or expired credentials were used.

34.8 Log Consistency Validation

Log consistency validation MUST ensure:

- All events exist in chronological order.
- No gaps or missing events.
- No duplicate EventIDs.
- No modified or deleted log entries.
- All events contain valid S3F payloads.

34.9 Compliance Failure Handling

If any compliance check fails:

- The asset MUST be marked NonCompliant.
- The system MUST enter SafeMode for that asset.
- A ComplianceFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No further lifecycle events MAY be processed until remediation.

34.10 Compliance Certification

Assets MAY be marked as SDLP-Certified if:

- All compliance stages pass successfully.
- No compliance failures exist in the assets history.
- Distributor credentials were valid for all events.
- All logs are complete and verifiable.

Certification MUST be recorded as a ComplianceCertified event.

34.11 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system compliance verification
- Standardized compliance reports in S3F
- Forward compatibility with new compliance rules
- Backward compatibility with legacy assets

35. OBJECT IDENTITY, IDS, AND BINDING RULES

35.1 Purpose

This section defines the identity model for SDLP assets, including the construction, uniqueness, persistence, and binding rules for all SDLP identifiers. SDLP IDs ensure that every digital asset, its lifecycle events, and its lineage can be unambiguously tracked across all SDLP-compliant systems.

35.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP-Registered Assets
- All SDLP Lifecycle Event Handlers
- All SDLP Verification and Compliance Systems

35.3 SDLP Identity Model Overview

Every SDLP asset MUST be assigned a globally unique SDLP ID at the moment of issuance. An SDLP ID is composed of the following fields:

DistributorID
CustomerID
ProductID
DownloadID
Date

Time

These fields, when concatenated in the SDLP-Standard Serialization Format (S3F), form the authoritative identity of the asset.

35.4 Identifier Field Definitions

The following fields MUST be present and MUST conform to SDLP rules:

DistributorID
: Unique identifier assigned to each SDLP-registered distributor.

CustomerID
: Unique identifier representing the customer or recipient.

ProductID
: Identifier representing the specific product or asset class.

DownloadID
: Unique identifier for the issuance instance.

Date
: RFC 3339 date of issuance.

Time
: RFC 3339 time of issuance.

35.5 Uniqueness Requirements

SDLP IDs MUST be globally unique. The following rules apply:

- No two assets may share the same SDLP ID.
- DistributorID MUST be unique across all distributors.
- CustomerID MUST be unique within the distributors namespace.
- ProductID MUST uniquely identify the product class.
- DownloadID MUST be unique for each issuance event.
- Date and Time MUST reflect the actual issuance timestamp.

Any collision MUST be treated as a critical compliance failure.

35.6 Persistence Requirements

SDLP IDs are immutable. The following rules apply:

- SDLP IDs MUST NOT change after issuance.
- Derived or copied assets MUST receive new SDLP IDs.
- ParentAssetID MUST be recorded for all derived assets.
- SDLP IDs MUST persist across transfers, archival, and recovery.

35.7 Binding Rules

The SDLP ID MUST be cryptographically bound to:

- The assets binary content (via HashPrimary and HashSecondary)
- The assets metadata

36. SDLP-ASSET CONTENT HASHING AND MULTI-HASH VERIFICATION

36.1 Purpose

This section defines the hashing, multi-hash verification, and content-binding requirements for all SDLP-registered assets. These rules ensure that every assets binary content is uniquely, immutably, and cryptographically tied to its SDLP identity and lifecycle history.

36.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines

36.3 Hashing Requirements

Every SDLP asset MUST include at least two independent cryptographic hashes:

HashPrimary
: The canonical hash used for identity binding.

HashSecondary
: A secondary hash using a different algorithm to provide cross-verification and algorithmic redundancy.

Both hashes MUST be computed over the exact binary content of the asset.

36.4 Approved Hash Algorithms

SDLP-compliant systems MUST support the following algorithms:

HashPrimary:

- SHA-256
- SHA-3-256

HashSecondary:

- BLAKE3
- SHA-512
- SHA-3-512

Additional algorithms MAY be approved in future SDLP revisions.

36.5 Hash Calculation Rules

Hashes MUST be calculated according to the following rules:

- Hashes MUST be computed on the raw binary content.
- No compression, encoding, or transformation MAY occur prior to hashing.
- Hashes MUST be stored in S3F using lowercase hexadecimal.
- Hashes MUST be recomputed for every lifecycle event involving content modification.

36.6 Multi-Hash Verification Requirements

Verification MUST include:

- Recomputing HashPrimary and HashSecondary.
- Comparing both values to the stored metadata.
- Rejecting the asset if either hash fails.
- Logging IntegrityCheckPerformed or IntegrityCheckFailed.

Verification MUST occur during:

- Asset issuance
- Asset transfer
- Asset duplication
- Compliance checks
- Audit operations

- Recovery operations

36.7 Hash Binding Requirements

HashPrimary and HashSecondary MUST be cryptographically bound to:

- The SDLP ID
- The assets metadata
- The lifecycle event that created or modified the asset

Any mismatch MUST invalidate the asset.

36.8 Hash Evolution and Algorithm Deprecation

SDLP supports algorithm evolution. The following rules apply:

- Deprecated algorithms MUST remain verifiable for legacy assets.
- New assets MUST use only approved algorithms.
- Distributors MUST migrate to new algorithms within the transition window defined by SDLP governance.
- Multi-hash verification MUST continue to function across algorithm generations.

36.9 Failure Handling

If any hash verification fails:

- The asset MUST be marked Invalid.
- A ComplianceFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

36.10 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system hash verification
- Multi-hash comparison across algorithm families
- Forward compatibility with new hashing algorithms
- Backward compatibility with legacy hash formats

37. SDLP-ASSET LINEAGE, DERIVATION, AND TRANSFORMATION RULES

37.1 Purpose

This section defines the rules governing asset lineage, derivation, transformation, and inheritance within the SDLP ecosystem. These rules ensure that every assets origin, parentage, and transformation history can be reconstructed with complete accuracy across all SDLP-compliant systems.

37.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP-Registered Assets
- All SDLP Lifecycle Event Handlers
- All SDLP Verification and Compliance Systems

37.3 Lineage Model Overview

SDLP defines lineage as the complete ancestry of an asset, including all parent, child, and sibling relationships. Lineage MUST be traceable through:

- ParentAssetID

- Lifecycle events
- Event Log entries
- HashPrimary and HashSecondary bindings
- SDLP ID relationships

Lineage MUST remain intact across all systems and transfers.

37.4 Asset Derivation Rules

An asset is considered derived when it is created from an existing asset through transformation, extraction, or modification. The following rules apply:

- A derived asset MUST receive a new SDLP ID.
- ParentAssetID MUST reference the original asset.
- The derivation MUST be recorded as an AssetDerived event.
- The derived asset MUST NOT inherit the parents DownloadID.
- The derived asset MUST compute new HashPrimary and HashSecondary.

37.5 Asset Duplication Rules

Duplication occurs when an asset is copied without modification. The following rules apply:

- A duplicated asset MUST receive a new SDLP ID.
- ParentAssetID MUST reference the original asset.
- The duplication MUST be recorded as an AssetCopied event.
- HashPrimary and HashSecondary MUST match the parent.
- Metadata MUST reflect the duplication event.

37.6 Asset Transformation Rules

Transformation occurs when an assets binary content changes. This includes format conversion, compression, editing, or structural modification. The following rules apply:

- A transformed asset MUST receive a new SDLP ID.
- ParentAssetID MUST reference the original asset.
- HashPrimary and HashSecondary MUST be recomputed.
- The transformation MUST be recorded as an AssetTransformed event.
- The transformed asset MUST NOT inherit the parents hashes.

37.7 Multi-Parent Derivation

Some assets may be derived from multiple parent assets (e.g., compilations, merged documents, composite media). The following rules apply:

- MultiParentIDs MUST list all contributing ParentAssetIDs.
- The derivation MUST be recorded as an AssetComposite event.
- The composite asset MUST compute new hashes.
- All parent assets MUST be valid and compliant.

37.8 Lineage Integrity Requirements

SDLP-compliant systems MUST ensure:

- ParentAssetID references a valid, existing asset.
- No circular lineage relationships exist.
- Lineage chains are complete and unbroken.
- All lineage events exist in the Event Log.
- Lineage can be reconstructed deterministically.

37.9 Lineage Verification

Verification MUST include:

- Validating ParentAssetID or MultiParentIDs.
- Ensuring all referenced assets exist and are compliant.
- Confirming chronological ordering of lineage events.
- Verifying hash relationships for duplicated assets.
- Ensuring transformed assets have new hashes.

37.10 Failure Handling

If lineage verification fails:

- The asset MUST be marked NonCompliant.
- A LineageFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

37.11 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system lineage reconstruction
- Multi-parent lineage models
- Forward compatibility with new lineage event types
- Backward compatibility with legacy lineage structures

38. SDLP-POLICY STRUCTURE, VERSIONING, AND ENFORCEMENT PHYSICS

38.1 Purpose

This section defines the structure, versioning model, and enforcement physics of SDLP policies. SDLP policies govern the behavior of distributors, assets, lifecycle events, and verification systems. Enforcement physics describe how policies propagate, bind, and constrain SDLP-compliant systems.

38.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification and Compliance Systems
- All SDLP Governance and Policy Engines

38.3 Policy Structure

All SDLP policies MUST follow the SDLP Policy Structure Model (SPSM), which consists of:

PolicyID
: Unique identifier for the policy.

PolicyClass
: One of: Core, Security, Compliance, Lifecycle, Metadata, Identity, Hashing, Logging, Governance.

PolicyVersion
: Semantic version number (MAJOR.MINOR.PATCH).

PolicyScope
: Defines which SDLP components the policy applies to.

PolicyRules
: A structured list of normative requirements.

EnforcementLevel
: One of: Advisory, Required, Mandatory, Critical.

EffectiveDate
: RFC 3339 timestamp when the policy becomes active.

DeprecationDate
: Optional timestamp for scheduled retirement.

PolicySignature
: Cryptographic signature issued by SDLP Governance.

38.4 Policy Versioning Model

SDLP uses semantic versioning for all policies:

MAJOR
: Introduces breaking changes or new mandatory requirements.

MINOR
: Adds new optional rules or clarifications.

PATCH
: Fixes errors, typos, or non-breaking clarifications.

The following rules apply:

- MAJOR updates MUST trigger compliance revalidation.
- MINOR updates MUST be supported by all new assets.
- PATCH updates MAY be adopted immediately without revalidation.

38.5 Policy Binding Rules

Policies MUST bind to:

- SDLP IDs
- Lifecycle events
- Metadata structures
- Hashing algorithms
- Distributor authentication sessions
- Event Log structures

Binding MUST ensure:

- Policies cannot be bypassed.
- Policies cannot be selectively applied.
- Policies remain attached to assets across transfers.

38.6 Enforcement Physics

Enforcement physics define how policies exert force across SDLP systems. Enforcement MUST follow these principles:

Deterministic Enforcement
: Policy outcomes MUST be predictable and reproducible.

Non-Bypassability
: No SDLP component may circumvent policy rules.

Propagation
: Policies MUST propagate to all dependent systems.

Temporal Consistency
: Policies MUST apply based on EffectiveDate and event timestamps.

Immutable Enforcement History
: Enforcement decisions MUST be logged and immutable.

38.7 Enforcement Levels

EnforcementLevel determines the severity of policy requirements:

Advisory
: Recommendations; non-binding.

Required
: Must be followed unless overridden by a higher-level policy.

Mandatory
: Must be followed; violations trigger compliance failures.

Critical
: Must be followed; violations trigger SafeMode and halt all lifecycle events.

38.8 Policy Evaluation Rules

SDLP-compliant systems MUST evaluate policies:

- At asset issuance
- At every lifecycle event
- During compliance checks
- During audit operations
- During distributor authentication

Evaluation MUST include:

- Version compatibility checks
- EnforcementLevel resolution
- Timestamp alignment
- Signature verification

38.9 Policy Conflict Resolution

If multiple policies apply simultaneously:

- Critical overrides Mandatory
- Mandatory overrides Required
- Required overrides Advisory
- Higher MAJOR version overrides lower MAJOR version
- If versions match, higher MINOR overrides lower MINOR
- If MINOR matches, higher PATCH overrides lower PATCH

Conflicts MUST be logged as PolicyConflict events.

38.10 Policy Deprecation and Retirement

Policies MAY be deprecated. The following rules apply:

- Deprecated policies MUST remain enforceable until retirement.
- Retirement MUST be announced at least 180 days in advance.
- Assets created before retirement MAY continue using the old policy if allowed by SDLP governance.

38.11 Failure Handling

If policy enforcement fails:

- The system MUST enter SafeMode.
- A PolicyEnforcementFailure event MUST be logged.

- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

38.12 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system policy synchronization
- Policy signature verification
- Forward compatibility with new policy classes
- Backward compatibility with legacy policy versions

39. DECAY MECHANICS AND USE-BASED LIFECYCLE PHYSICS

39.1 Purpose

This section defines the decay mechanics, predictive lifecycle physics, and use-based degradation rules that govern SDLP-registered assets. These mechanics ensure that asset state, integrity, and lifecycle transitions follow deterministic, measurable, and verifiable rules based on usage, time, and environmental factors.

39.2 Scope

These requirements apply to:
- All SDLP-Registered Assets
- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance Systems

39.3 Decay Model Overview

SDLP defines decay as the measurable reduction in asset stability, integrity, or compliance probability over time or use. Decay is governed by:

- Time-Based Decay (TBD)
- Use-Based Decay (UBD)
- Event-Driven Decay (EDD)
- Environmental Decay Modifiers (EDM)

Decay MUST be predictable, quantifiable, and reconstructable.

39.4 Time-Based Decay (TBD)

TBD applies to all assets regardless of use. The following rules apply:

- TBD begins at TimestampIssued.
- TBD increases monotonically with real time.
- TBD MUST NOT regress.
- TBD MUST be recorded in the assets metadata as DecayTimeIndex.

TBD MAY trigger lifecycle transitions such as ArchiveEligible.

39.5 Use-Based Decay (UBD)

UBD measures decay caused by asset usage. Usage includes:

- Access events
- Playback events
- Execution events
- Read/write operations
- Verification operations

UBD MUST:

- Increase with each usage event
- Be recorded as DecayUseIndex
- Be included in all compliance checks
- Influence predictive lifecycle physics

39.6 Event-Driven Decay (EDD)

Certain lifecycle events accelerate decay. These include:

- AssetCopied
- AssetTransformed
- AssetRecovered
- IntegrityCheckFailed

EDD MUST:

- Increase DecayEventIndex
- Be logged as part of the event payload
- Influence lifecycle predictions

39.7 Environmental Decay Modifiers (EDM)

EDM adjusts decay based on environmental factors such as:

- Storage medium reliability
- Distributor infrastructure quality
- Network integrity
- Cryptographic algorithm age

EDM MUST be:

- Deterministic
- Documented in S3F
- Applied consistently across systems

39.8 Predictive Lifecycle Physics

SDLP-compliant systems MUST compute a PredictiveLifecycleScore (PLS) using:

```
PLS = f(TBD, UBD, EDD, EDM)
```

PLS MUST:

- Predict the probability of future compliance
- Influence lifecycle transitions
- Be included in compliance reports
- Be recalculated at every lifecycle event

39.9 Lifecycle Transition Triggers

The following transitions MAY be triggered by decay thresholds:

- ArchiveEligible
- RecoveryRequired
- IntegrityCheckRequired
- ComplianceReviewRequired
- RetirementRecommended

Thresholds MUST be defined by SDLP governance.

39.10 Rewind Rule

If an asset is restored from a previous valid state:

- TBD MUST remain unchanged.
- UBD MUST remain unchanged.
- EDD MUST increase by RewindPenaltyIndex.
- A RewindEvent MUST be logged.
- The restored state MUST pass full integrity verification.

Rewind MUST NOT reset decay.

39.11 Acceleration Rule

If an asset experiences repeated failures or anomalies:

- Decay indices MUST accelerate according to SDLP rules.
- Acceleration MUST be recorded as DecayAccelerationIndex.
- A DecayAcceleration event MUST be logged.
- PredictiveLifecycleScore MUST be recalculated immediately.

39.12 Verification Requirements

Verification MUST confirm:

- All decay indices are present and non-regressive.
- PLS is correctly computed.
- Decay thresholds match lifecycle transitions.
- Rewind and Acceleration rules were applied correctly.

39.13 Failure Handling

If decay verification fails:

- The asset MUST be marked NonCompliant.
- A DecayVerificationFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

39.14 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system decay reconstruction
- Deterministic PLS computation
- Forward compatibility with new decay indices
- Backward compatibility with legacy decay models

40. SDLP-ASSET STABILITY INDEX AND HEALTH STATE MODEL

40.1 Purpose

This section defines the SDLP Asset Stability Index (ASI) and the Asset Health State Model (AHSM). These mechanisms provide a unified, deterministic framework for evaluating the operational stability, compliance readiness, and lifecycle viability of SDLP-registered assets.

40.2 Scope

These requirements apply to:

- All SDLP-Registered Assets
- All SDLP Distributors
- All SDLP Verification Engines
- All SDLP Lifecycle Event Handlers
- All SDLP Compliance Systems

40.3 Asset Stability Index (ASI) Overview

ASI is a composite numerical score representing the current stability, reliability, and compliance probability of an asset. ASI MUST be computed using:

```
ASI = g(DecayTimeIndex, DecayUseIndex, DecayEventIndex,
        DecayAccelerationIndex, PredictiveLifecycleScore,
        IntegrityHistory, RecoveryHistory)
```

ASI MUST be:

- Deterministic
- Non-negative
- Recomputable from logs and metadata
- Included in all compliance reports

40.4 ASI Input Components

ASI MUST incorporate the following components:

DecayTimeIndex
Time-based decay contribution.

DecayUseIndex
Use-based decay contribution.

DecayEventIndex
Event-driven decay contribution.

DecayAccelerationIndex
Acceleration due to repeated failures.

PredictiveLifecycleScore
Probability of future compliance.

IntegrityHistory
Count and severity of past integrity failures.

RecoveryHistory
Count and severity of past recovery events.

40.5 ASI Calculation Rules

ASI MUST follow these rules:

- ASI MUST decrease as decay indices increase.
- ASI MUST decrease after integrity failures.
- ASI MUST decrease after recovery events.
- ASI MUST NOT increase unless:
  - A successful recovery event occurs, or
  - A policy-defined stabilization event occurs.

- ASI MUST be recalculated:
  - At every lifecycle event
  - During compliance checks
  - During audit operations
  - After any decay index change

40.6 Asset Health State Model (AHSM)

AHSM defines discrete health states for SDLP assets. The following states MUST be supported:

Healthy
ASI above the HealthyThreshold. No recent failures.

Stable
: ASI above the StableThreshold. Minor decay or minor historical failures.

Degraded
: ASI below StableThreshold but above CriticalThreshold. Significant decay or repeated failures.

Critical
: ASI below CriticalThreshold. High risk of non-compliance.

NonCompliant
: Asset has failed verification or exceeded decay limits.

40.7 Health State Transition Rules

Transitions MUST follow deterministic rules:

- Healthy ? Stable
  Triggered by moderate decay or minor failures.

- Stable ? Degraded
  Triggered by increased decay or repeated failures.

- Degraded ? Critical
  Triggered by severe decay or integrity failures.

- Critical ? NonCompliant
  Triggered by compliance failure or threshold breach.

- Any State ? Healthy
  Only allowed after a successful RecoveryEvent and full integrity verification.

40.8 Threshold Definitions

Thresholds MUST be defined by SDLP governance:

HealthyThreshold
StableThreshold
CriticalThreshold

Thresholds MUST be:

- Documented in S3F
- Versioned
- Applied consistently across systems

40.9 Verification Requirements

Verification MUST confirm:

- ASI is correctly computed.
- Health state matches ASI thresholds.
- All transitions follow AHSM rules.
- No unauthorized state changes occurred.
- All decay indices and history logs are consistent.

40.10 Failure Handling

If ASI or health state verification fails:

- The asset MUST be marked NonCompliant.
- A HealthStateVerificationFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

40.11 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system ASI reconstruction
- Deterministic health state evaluation
- Forward compatibility with new health states
- Backward compatibility with legacy ASI models

41. SDLP-ASSET ENTROPY MODEL AND INFORMATION DISPERSION RULES

41.1 Purpose

This section defines the SDLP Entropy Model (SEM) and the rules governing information dispersion, disorder accumulation, and structural drift within SDLP-registered assets. Entropy represents the measurable tendency of an asset to accumulate disorder over time and use, influencing stability, compliance probability, and lifecycle transitions.

41.2 Scope

These requirements apply to:

- All SDLP-Registered Assets
- All SDLP Distributors
- All SDLP Verification Engines
- All SDLP Lifecycle Event Handlers
- All SDLP Compliance Systems

41.3 Entropy Model Overview

SDLP defines entropy as a quantifiable measure of structural, informational, or operational disorder. Entropy MUST be computed using:

```
EntropyIndex = h(DecayIndices, ASI, IntegrityHistory,
                 TransformationHistory, EnvironmentalFactors)
```

EntropyIndex MUST be:

- Deterministic
- Non-negative
- Recomputable from logs and metadata
- Included in compliance and audit reports

41.4 Sources of Entropy

Entropy arises from the following sources:

Structural Entropy
: Drift in metadata, formatting, or structural consistency.

Operational Entropy
: Disorder introduced through repeated usage or access.

Transformational Entropy
: Disorder introduced by transformations, rewrites, or format conversions.

Environmental Entropy
: Disorder caused by storage medium degradation, network inconsistencies, or distributor infrastructure variance.

41.5 Entropy Accumulation Rules

Entropy MUST accumulate according to the following rules:

- EntropyIndex MUST increase with decay indices.
- EntropyIndex MUST increase after transformations.
- EntropyIndex MUST increase after recovery events.
- EntropyIndex MUST increase after integrity failures.
- EntropyIndex MUST NOT decrease unless:
  A successful stabilization event occurs, or
  A policy-defined entropy reset event occurs.

41.6 Entropy Thresholds

SDLP governance MUST define the following thresholds:

EntropyStableThreshold
EntropyDegradedThreshold
EntropyCriticalThreshold

Thresholds MUST be:
- Documented in S3F
- Versioned
- Applied consistently across systems

41.7 Entropy-Driven Lifecycle Effects

When EntropyIndex exceeds thresholds, the following transitions MAY be triggered:

- StabilityReviewRequired
- IntegrityCheckRequired
- RecoveryRecommended
- ArchiveRecommended
- RetirementRequired

These transitions MUST be logged as EntropyTransition events.

41.8 Entropy Dispersion Rules

Entropy dispersion describes how disorder propagates across related assets. The following rules apply:

- Parent entropy MUST NOT automatically propagate to children.
- Child entropy MUST NOT retroactively affect parents.
- Composite assets MUST compute entropy as a function of all contributing parents.
- Entropy dispersion MUST be deterministic and documented.

41.9 Entropy Reset and Stabilization Events

Certain events MAY reduce entropy:

StabilizationEvent
: A policy-defined event that reduces entropy after successful verification and structural normalization.

RecoveryEvent
: MAY reduce entropy if the recovered state is demonstrably more stable than the previous state.

Entropy resets MUST be:
- Logged as EntropyReset events
- Verified through full integrity checks

41.10 Verification Requirements

Verification MUST confirm:

- EntropyIndex is correctly computed.
- Entropy thresholds match lifecycle transitions.
- No unauthorized entropy reductions occurred.
- Entropy dispersion rules were applied correctly.

42. SDLP-LIFECYCLE PHYSICS, DECAY MODEL, AND TERMINAL TRANSITION RULES

42.1 Purpose

This section defines the lifecycle physics governing SDLP instances, including decay progression, entropy accumulation, ASI degradation, P-value computation, hospice transitions, and the mandatory conditions under which an instance MUST undergo Bit-Drop. These physics ensure deterministic lifecycle behavior across all SDLP-compliant systems.

42.2 Scope

These requirements apply to:
- All SDLP instances
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance Systems
- All SDLP Distributors

42.3 Lifecycle Physics Overview

SDLP lifecycle physics define the mathematical and structural rules governing the evolution, degradation, and termination of an instance. Lifecycle physics include:

- DecayIndex progression
- EntropyIndex accumulation
- ASI degradation
- PFloat and PInteger computation
- Hospice transitions
- Terminal transitions
- Bit-Drop as the irreversible destruction operator

These physics MUST be deterministic, reproducible, and independent of implementation details.

42.4 Decay Model

The DecayIndex represents the structural degradation of an instance over time or under stress. Systems MUST:

- Increment DecayIndex deterministically
- Recompute decay after every lifecycle event
- Reject negative or non-monotonic decay values
- Trigger Hospice when DecayIndex = DecayCriticalThreshold

Decay MUST NOT be reset except during authorized reconstruction.

42.5 Entropy Model

The EntropyIndex represents accumulated disorder within the instance. Systems MUST:

- Increase entropy upon corruption, tampering, or invalid states
- Trigger SafeMode when EntropyIndex = EntropyCriticalThreshold
- Prevent lifecycle transitions while entropy is critical
- Recompute entropy during audits and verification

Entropy MUST be monotonic unless corrected through validated remediation.

42.6 ASI (Asset Stability Index)

ASI represents the structural stability of an instance. Systems MUST:

- Recompute ASI after every lifecycle event
- Trigger Hospice when ASI = ASICriticalThreshold
- Prevent activation if ASI is below initialization thresholds

ASI MUST be derived from decay, entropy, and structural integrity metrics.

42.7 P-Value Computation

PFloat and PInteger represent the lifecycle exhaustion state of an instance.

- PFloat is a continuous exhaustion metric.
- PInteger is the discrete exhaustion boundary.

Systems MUST:

- Recompute PFloat and PInteger after every lifecycle event
- Trigger Hospice when PFloat = PHospiceThreshold
- Trigger Bit-Drop when PInteger = 1

PInteger MUST be treated as authoritative for terminal transitions.

42.8 Hospice Transition Rules

Hospice is a mandatory pre-terminal state. An instance MUST enter Hospice when:

- ASI = ASICriticalThreshold
- DecayIndex = DecayCriticalThreshold
- PFloat = PHospiceThreshold
- EntropyIndex = EntropyCriticalThreshold (post-SafeMode)

Hospice prohibits:

- Transfers
- Transformations
- Duplication
- Policy overrides
- Any operation that increases structural risk

Hospice MUST precede Bit-Drop unless Bit-Drop is triggered by catastrophic failure.

42.9 Unnatural Death Conditions

Unnatural death occurs when an instance MUST undergo immediate Bit-Drop due to:

- Catastrophic corruption
- Tamper detection
- Unauthorized manipulation
- Policy violations requiring destruction
- Failed SafeMode recovery
- Failed initialization trust evaluation

Unnatural death MUST be instantaneous and MUST NOT allow intermediate lifecycle states.

42.10 Bit-Drop: Canonical Destruction Operator

Bit-Drop is the irreversible, instantaneous destruction of an SDLP instance. Bit-Drop MUST:

- Occur in zero computational time from the instances perspective
- Zeroize all binary content atomically
- Produce a final immutable destruction log entry
- Terminate all lifecycle continuity
- Prevent any further transitions or operations

Bit-Drop MUST NOT be implemented as a rate, process, or gradual degradation.

42.11 Bit-Drop Triggers

An instance MUST undergo Bit-Drop when:

- PInteger = 1
- Hospice recovery fails
- SafeMode recovery fails
- Initialization trust evaluation fails
- Catastrophic corruption is detected
- Policy mandates destruction
- Governance mandates destruction

Bit-Drop MUST be treated as a mandatory, non-overridable transition.

42.12 Post-Destruction Semantics

After Bit-Drop:

- The instance is considered non-existent
- No lifecycle events may be appended
- No reconstruction or rehydration is permitted
- No lineage continuation is permitted
- No reinstantiation is permitted

Any attempt to interact with a Bit-Dropped instance MUST be rejected as a lifecycle violation.

42.13 Rationale

Lifecycle physics ensure deterministic behavior, prevent corruption, and enforce predictable terminal transitions. Bit-Drop provides the final safeguard against unauthorized use, corruption, or lifecycle inconsistency, ensuring that SDLP instances enforce their own boundaries without reliance on human discretion.

43. SDLP-EVENT TYPE REGISTRY AND STANDARDIZED EVENT DEFINITIONS

43.1 Purpose

This section defines the authoritative registry of SDLP event types and the normative requirements for their structure, semantics, and usage. Event types provide the universal vocabulary for describing all lifecycle, compliance, integrity, security, and governance actions within SDLP-compliant systems.

43.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance Systems
- All SDLP Asset Registries

43.3 Event Type Registry Overview

The SDLP Event Type Registry (SETR) is the canonical list of all event types recognized by SDLP. Each event type MUST be:

- Uniquely named
- Deterministically defined
- Version-stable
- Backward compatible
- Forward extensible

Event types MUST NOT be redefined once published. New event types MUST be introduced through additive versioning.

43.4 Event Type Structure

Each event type MUST include the following fields:

EventType
The canonical name of the event.

EventClass
One of: Lifecycle, Integrity, Compliance, Security, Governance, System, Metadata, Physics.

EventVersion
Semantic version number (MAJOR.MINOR.PATCH).

RequiredFields
List of mandatory S3F fields.

OptionalFields
List of optional S3F fields.

Description
Human-readable explanation of the event.

Constraints
Normative rules governing event usage.

43.5 Standard Event Types

The following event types MUST be supported by all SDLP-compliant systems:

AssetCreated
AssetIssued
AssetTransferred
AssetCopied
AssetDerived
AssetTransformed
AssetArchived
AssetRetired

MetadataUpdated

IntegrityCheckPerformed
IntegrityCheckFailed

ComplianceCertified
ComplianceFailure

DistributorAuthentication
DistributorAuthorizationFailure

SessionClosed

SystemError
SystemRecovery

LineageFailure

PolicyConflict
PolicyEnforcementFailure

DecayAcceleration
DecayVerificationFailure

HealthStateVerificationFailure

EntropyTransition
EntropyReset
EntropyVerificationFailure

PValueTransition
PValueVerificationFailure

ASITransition
ASIVerificationFailure

HospiceEntered
HospiceExited

SafeModeEntered
SafeModeExited

BitDropInitiated
BitDropCompleted
BitDropVerificationFailure

43.6 Event Naming Rules

Event names MUST:

- Use PascalCase
- Contain only ASCII letters and digits
- Not exceed 64 characters
- Not include spaces, punctuation, or symbols
- Be descriptive but concise

43.7 Event Class Definitions

EventClass MUST be one of the following:

Lifecycle
: Events that modify or advance the asset lifecycle.

Integrity
: Events related to hashing, verification, or corruption.

Compliance
: Events related to compliance checks or failures.

Security
: Events related to authentication, authorization, or session management.

Governance
: Events related to policy enforcement or conflict resolution.

System

Events related to system errors, recovery, or SafeMode.

Metadata
: Events related to metadata creation or modification.

Physics
: Events related to decay, entropy, ASI, P-value transitions, hospice transitions, and Bit-Drop.

43.8 Event Payload Requirements

Each event MUST include:

- EventID (globally unique)
- Timestamp (RFC 3339)
- DistributorID
- AssetID (if applicable)
- EventPayload (S3F object)

EventPayload MUST contain all RequiredFields defined by the event type and MUST NOT contain prohibited fields.

43.9 Event Ordering Rules

SDLP-compliant systems MUST ensure:

- Events are strictly ordered by timestamp
- No retroactive insertion of events
- No modification of existing events
- Corrections are recorded as new events
- Event ordering is deterministic across distributed systems

43.10 Event Versioning Rules

Event types MUST follow semantic versioning:

MAJOR
: Breaking changes to event structure.

MINOR
: Additive, non-breaking changes.

PATCH
: Clarifications or corrections.

Systems MUST support all MAJOR versions active at the time of asset creation.

43.11 Event Deprecation Rules

Deprecated events MUST:

- Remain verifiable
- Remain interpretable
- Not be used for new assets
- Be documented in SETR

43.12 Failure Handling

If an event fails validation:

- The event MUST be rejected.
- A SystemError event MUST be logged.
- The asset MUST enter SafeMode.
- The distributor MUST be notified.

43.13 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system event interpretation
- Deterministic event ordering
- Forward compatibility with new event types
- Backward compatibility with legacy event types

44. SDLP-STANDARD SERIALIZATION FORMAT (S3F)

44.1 Purpose

This section defines the SDLP-Standard Serialization Format (S3F), the canonical encoding used for all SDLP metadata, event payloads, compliance reports, and policy objects. S3F ensures that all SDLP systems serialize and deserialize structured data in a deterministic, interoperable, and verifiable manner.

44.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance and Audit Systems
- All SDLP Governance and Policy Engines

44.3 S3F Design Principles

S3F MUST adhere to the following principles:

Determinism
The same input MUST always produce the same serialized output.

Canonical Ordering
Fields MUST appear in lexicographic ASCII order.

Strict Typing
Each field MUST declare and conform to its type.

Immutability
Serialized objects MUST NOT be altered after creation.

Interoperability
All SDLP systems MUST be able to parse S3F objects.

Auditability
S3F MUST support deterministic reconstruction of asset history and event payloads.

44.4 S3F Data Types

S3F supports the following primitive types:

String
UTF-8 encoded, ASCII-safe unless otherwise specified.

Integer
Base-10, non-negative, no leading zeros.

Float
Decimal notation with at least one digit before and after the decimal point.

Boolean
"true" or "false".

Timestamp
RFC 3339 format.

BinaryHash
Lowercase hexadecimal string.

Array
Ordered list of S3F values.

Object
Key-value map with lexicographically ordered keys.

44.5 Canonical Ordering Rules

All S3F objects MUST follow these ordering rules:

- Keys MUST be sorted in ascending ASCII order.
- Arrays MUST preserve insertion order.
- No duplicate keys are allowed.
- Optional fields MUST NOT appear before required fields unless lexicographic ordering requires it.

44.6 Field Naming Rules

Field names MUST:

- Use PascalCase
- Contain only ASCII letters and digits
- Not exceed 64 characters
- Not contain spaces, punctuation, or symbols

44.7 Required S3F Fields for Asset Metadata

All SDLP assets MUST include the following fields:

AssetID
DistributorID
CustomerID
ProductID
DownloadID
TimestampIssued
HashPrimary
HashSecondary
LifecycleState
ParentAssetID (if applicable)
MultiParentIDs (if applicable)

Optional fields MAY be included but MUST follow canonical ordering.

44.8 Required S3F Fields for Event Payloads

All SDLP events MUST include:

EventID
EventType
Timestamp
DistributorID
AssetID (if applicable)
Payload (Object)

Payload MUST contain all RequiredFields defined by the event type.

44.9 Serialization Rules

S3F serialization MUST follow these rules:

- UTF-8 encoding
- No BOM
- No trailing commas
- No extraneous whitespace
- Indentation using 4 spaces
- Newline termination using LF (0x0A)

Serialized output MUST be byte-for-byte identical across systems.

44.10 Deserialization Rules

S3F deserialization MUST:

- Reject malformed objects
- Reject duplicate keys
- Reject out-of-order keys
- Reject invalid types
- Reject missing required fields

Deserialization MUST NOT attempt to correct or infer missing data.

44.11 S3F Validation Requirements

Validation MUST confirm:

- Canonical ordering
- Correct data types
- Required fields present
- No prohibited fields
- Hash fields match expected formats
- Timestamps follow RFC 3339

Validation MUST occur during all lifecycle events.

44.12 S3F Versioning

S3F MUST include a version field:

```
S3FVersion
```

Versioning MUST follow semantic versioning:

```
MAJOR.MINOR.PATCH
```

Systems MUST support all MAJOR versions active at the time of asset creation.

44.13 Failure Handling

If S3F validation fails:

- The asset or event MUST be rejected.
- A SerializationFailure event MUST be logged.
- The system MUST enter SafeMode for the affected asset.
- The distributor MUST be notified immediately.

44.14 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system S3F parsing
- Deterministic serialization

- Forward compatibility with new S3F fields
- Backward compatibility with legacy S3F structures

45. SDLP-CANONICAL LIFECYCLE STATES AND STATE TRANSITION RULES

45.1 Purpose

This section defines the canonical lifecycle states for all SDLP-registered assets and the normative rules governing transitions between those states. These rules ensure deterministic, auditable, and tamper-evident lifecycle progression across all SDLP-compliant systems.

45.2 Scope

These requirements apply to:
- All SDLP-Registered Assets
- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance Systems

45.3 Canonical Lifecycle States

All SDLP assets MUST exist in exactly one of the following canonical lifecycle states:

Instantiated
: The asset has been created but not yet issued.

Active
: The asset is valid, compliant, and in normal operational use.

Restricted
: The asset is temporarily limited due to compliance, integrity, or policy conditions.

Hospice
: The asset is nearing end-of-life due to decay, entropy, P-value thresholds, or repeated failures.

Decayed
: The asset has exceeded allowable decay or entropy limits and is no longer considered operationally stable.

Bit-Dropped
: The asset has undergone irreversible, instantaneous Bit-Drop and no longer exists in binary form.

45.4 State Definitions

Instantiated
- AssetID assigned.
- Metadata created.
- Hashes computed.
- No usage permitted.
- No lineage relationships yet established.

Active
- Asset is fully valid and compliant.
- All lifecycle events permitted.
- ASI, P-value, and entropy within acceptable thresholds.

Restricted
- Asset is temporarily limited.
- Triggered by compliance failures, integrity anomalies,

policy conflicts, or distributor authorization issues.
- Only remediation events permitted.

Hospice
- Asset is approaching end-of-life.
- Triggered by:
    ASI = ASICriticalThreshold
    EntropyIndex = EntropyCriticalThreshold (post-SafeMode)
    DecayIndex = DecayCriticalThreshold
    PFloat = PHospiceThreshold
    PInteger = 5
- Only limited lifecycle events permitted.
- Bit-Drop may be imminent.

Decayed
- Asset has exceeded decay or entropy thresholds.
- Asset is no longer stable or compliant.
- Only recovery or Bit-Drop events permitted.

Bit-Dropped
- Asset has undergone instantaneous, irreversible Bit-Drop.
- All binary content has been zeroized atomically.
- Only metadata and lineage remain for audit.
- No further lifecycle events permitted.

45.5 Permitted State Transitions

The following transitions are permitted:

Instantiated ? Active

Active ? Restricted
Active ? Hospice
Active ? Decayed
Active ? Bit-Dropped (catastrophic failure)

Restricted ? Active
Restricted ? Hospice
Restricted ? Decayed
Restricted ? Bit-Dropped (catastrophic failure)

Hospice ? Decayed
Hospice ? Bit-Dropped

Decayed ? Bit-Dropped

The following transitions are permitted only after successful recovery:

Restricted ? Active
Hospice ? Active
Decayed ? Active

45.6 Prohibited State Transitions

The following transitions MUST NOT occur:

- Any state ? Instantiated
- Bit-Dropped ? Any state
- Decayed ? Hospice
- Bit-Dropped ? Active
- Bit-Dropped ? Restricted
- Bit-Dropped ? Decayed

Any attempt to perform a prohibited transition MUST be rejected and logged as a LifecycleViolation event.

45.7 Transition Validation Requirements

All state transitions MUST be validated by:

- Integrity verification
- Compliance verification
- Policy enforcement checks
- Session authentication and authorization
- Event Log consistency checks

Validation MUST occur before the transition is committed.

45.8 Transition Logging Requirements

Every state transition MUST generate a LifecycleTransition event containing:

- PreviousState
- NewState
- TransitionReason
- Timestamp
- DistributorID
- ValidationSummary

45.9 SafeMode Requirements

If a transition fails validation:

- The asset MUST enter Restricted state.
- A LifecycleTransitionFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No further transitions MAY occur until remediation.

45.10 End-of-Life Transition Rules

The following rules apply:

- PInteger = 1 MUST trigger Bit-Drop.
- Excessive entropy MUST trigger transition to Decayed.
- Excessive decay MUST trigger transition to Decayed.
- Repeated integrity failures MUST trigger transition to Hospice.

45.11 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system lifecycle reconstruction
- Deterministic state evaluation
- Forward compatibility with new lifecycle states
- Backward compatibility with legacy lifecycle models

46. SDLP-SAFEMODE ARCHITECTURE AND SYSTEM QUARANTINE RULES

46.1 Purpose

This section defines the SDLP SafeMode architecture, including the rules, triggers, operational constraints, and recovery pathways for assets and systems entering SafeMode. SafeMode provides a controlled, restricted operational state designed to prevent further corruption, unauthorized actions, or compliance violations.

46.2 Scope

These requirements apply to:

- All SDLP-Registered Assets

- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance and Audit Systems

46.3 SafeMode Overview

SafeMode is a mandatory protective state activated when an asset or system encounters conditions that threaten integrity, compliance, or lifecycle correctness. SafeMode MUST:

- Restrict all non-remediation operations
- Preserve the current state for audit
- Prevent further lifecycle transitions
- Enforce heightened verification requirements
- Log all SafeMode actions

46.4 SafeMode Triggers

The following conditions MUST trigger SafeMode:

Integrity Failures
- Hash mismatch
- Signature mismatch
- Corrupted metadata

Compliance Failures
- Failed compliance verification
- Missing required events
- Invalid lineage references

Policy Violations
- PolicyEnforcementFailure
- PolicyConflict involving Mandatory or Critical policies

Security Violations
- Unauthorized distributor action
- Session hijacking or replay detection
- Credential revocation during active session

Physics Thresholds
- EntropyIndex = EntropyCriticalThreshold
- ASI below CriticalThreshold
- PInteger = 1 (prior to bitdump)

System Failures
- Logging subsystem failure
- Storage corruption
- Registry inconsistency

46.5 SafeMode Operational Constraints

While in SafeMode, the following restrictions apply:

- No lifecycle transitions except RecoveryEvent
- No metadata modifications except remediation fields
- No distributor operations except those explicitly authorized
- No asset issuance, transfer, duplication, or transformation
- No policy overrides
- No hash recalculation except during recovery
- No P-value recalculation except during recovery

All attempted prohibited actions MUST be rejected and logged as SafeModeViolation events.

46.6 SafeMode Logging Requirements

Upon entering SafeMode, the system MUST log:

SafeModeEntered
- Reason
- Triggering event
- Timestamp
- DistributorID (if applicable)

Upon exiting SafeMode, the system MUST log:

SafeModeExited
- Recovery method
- Validation summary
- Timestamp

46.7 Quarantine Rules

Assets in SafeMode are considered quarantined. Quarantine MUST:

- Isolate the asset from normal lifecycle operations
- Prevent cross-asset contamination
- Prevent lineage propagation
- Prevent composite asset creation involving quarantined assets

Quarantine MUST remain active until recovery is complete.

46.8 Recovery Pathways

The following recovery pathways are permitted:

Structural Recovery
- Metadata normalization
- S3F correction
- Rebuilding missing fields

Integrity Recovery
- Recomputing hashes
- Restoring from last valid state

Compliance Recovery
- Reconstructing missing events
- Resolving lineage inconsistencies

Security Recovery
- Reauthentication
- Session regeneration
- Credential replacement

Recovery MUST be validated before SafeMode can be exited.

46.9 Recovery Validation Requirements

Before exiting SafeMode, the system MUST verify:

- All integrity checks pass
- All compliance checks pass
- All policy conflicts are resolved
- All decay and entropy indices are consistent
- PFloat and PInteger are recomputed
- ASI is recalculated and above CriticalThreshold
- No prohibited transitions occurred during SafeMode

46.10 Automatic Transition Rules

The following transitions MUST occur automatically:

- If recovery fails ? asset transitions to Decayed
- If PInteger = 1 after recovery ? asset transitions to Bitdumped
- If entropy remains above critical threshold ? asset remains in SafeMode

46.11 Failure Handling

If SafeMode operations fail:

- The asset MUST be marked NonCompliant
- A SafeModeFailure event MUST be logged
- The distributor MUST be notified immediately
- The system MUST escalate to GovernanceReviewRequired

46.12 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system SafeMode reconstruction
- Deterministic quarantine behavior
- Forward compatibility with new SafeMode triggers
- Backward compatibility with legacy SafeMode models

47. SDLP-POLICY ENGINE, RUNTIME ENFORCEMENT, AND THRESHOLD GOVERNANCE

47.1 Purpose

This section defines the SDLP Policy Engine (SPE), including the structure, evaluation rules, runtime enforcement model, and threshold-based governance mechanisms that regulate all SDLP operations. Policies define the mandatory behavioral constraints for distributors, assets, lifecycle events, and verification systems.

47.2 Scope

These requirements apply to:
- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance Systems
- All SDLP Governance and Policy Engines

47.3 Policy Object Structure

All SDLP policies MUST be represented as canonical S3F objects with the following fields:

PolicyID
: Globally unique identifier.

PolicyClass
: One of: Core, Security, Compliance, Lifecycle, Metadata, Identity, Hashing, Logging, Physics, Governance.

PolicyVersion
: Semantic version number (MAJOR.MINOR.PATCH).

EnforcementLevel
: One of: Advisory, Required, Mandatory, Critical.

Thresholds
: Object containing numeric or boolean enforcement thresholds.

Rules

Ordered list of normative requirements.

EffectiveDate
RFC 3339 timestamp when the policy becomes active.

DeprecationDate
Optional timestamp for scheduled retirement.

PolicySignature
Cryptographic signature issued by SDLP Governance.

47.4 Policy Evaluation Model

The SDLP Policy Engine MUST evaluate policies:

- At asset issuance
- At every lifecycle event
- During compliance checks
- During audit operations
- During distributor authentication
- During SafeMode entry and exit

Evaluation MUST include:

- Signature verification
- Version compatibility checks
- EnforcementLevel resolution
- Threshold comparison
- Rule validation
- Temporal alignment with EffectiveDate

47.5 Threshold Enforcement Rules

Policies MAY define thresholds that MUST be enforced at runtime.
Thresholds MAY include:

- ASI thresholds
- Entropy thresholds
- Decay thresholds
- P-value thresholds
- Rate limits
- Session constraints
- Hash algorithm requirements
- Metadata completeness requirements

Thresholds MUST be:

- Deterministic
- Non-overridable
- Logged upon violation
- Evaluated before lifecycle transitions

47.6 Runtime Enforcement Physics

Runtime enforcement MUST follow these principles:

Deterministic Enforcement
The same inputs MUST always produce the same enforcement outcome.

Non-Bypassability
No SDLP component may circumvent policy rules.

Temporal Consistency
Policies MUST apply based on EffectiveDate and event timestamps.

Immutable Enforcement History
: All enforcement decisions MUST be logged.

Cascading Enforcement
: Violations MAY trigger additional enforcement actions such as SafeMode, Hospice transition, or Bitdump.

47.7 Enforcement Actions

When a policy rule or threshold is violated, the Policy Engine MUST take one or more of the following actions:

- Reject the lifecycle event
- Enter SafeMode
- Transition the asset to Restricted or Hospice
- Trigger ComplianceReviewRequired
- Trigger IntegrityCheckRequired
- Trigger Bitdump (if PInteger = 1)
- Notify the distributor
- Log a PolicyEnforcementFailure event

47.8 Policy Conflict Resolution

If multiple policies apply simultaneously, the following precedence rules MUST be used:

- Critical overrides Mandatory
- Mandatory overrides Required
- Required overrides Advisory
- Higher MAJOR version overrides lower MAJOR version
- If MAJOR matches, higher MINOR overrides lower MINOR
- If MINOR matches, higher PATCH overrides lower PATCH

Conflicts MUST be logged as PolicyConflict events.

47.9 Policy Propagation Rules

Policies MUST propagate across:

- Asset transfers
- Distributor boundaries
- Verification engines
- Compliance systems
- Multi-system registries

Propagation MUST be deterministic and cryptographically verifiable.

47.10 Policy Violation Logging Requirements

Every policy violation MUST generate a PolicyEnforcementFailure event containing:

- PolicyID
- PolicyVersion
- EnforcementLevel
- Threshold breached (if applicable)
- Rule violated
- Timestamp
- DistributorID
- AssetID (if applicable)
- EnforcementAction taken

47.11 Governance Review Requirements

If repeated policy violations occur:

- The asset MUST be flagged for GovernanceReviewRequired.
- The distributor MAY be temporarily restricted.
- The system MAY require re-authentication.
- Additional policies MAY be applied automatically.

47.12 Failure Handling

If policy evaluation or enforcement fails:

- The system MUST enter SafeMode.
- A PolicyEngineFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

47.13 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system policy synchronization
- Deterministic policy evaluation
- Forward compatibility with new policy classes
- Backward compatibility with legacy policy versions

48. SDLP-LOGGING ARCHITECTURE AND IMMUTABLE EVENT RECORDS

48.1 Purpose

This section defines the SDLP Logging Architecture (SLA), including the structure, retention rules, immutability guarantees, and cross-system synchronization requirements for all SDLP event logs. Logging is the authoritative source of truth for asset history, compliance verification, lineage reconstruction, and auditability.

48.2 Scope

These requirements apply to:
- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance and Audit Systems
- All SDLP Governance Systems

48.3 Logging Model Overview

SDLP uses an append-only, immutable logging model. Logs MUST:
- Record every lifecycle, compliance, integrity, and policy event
- Preserve strict chronological ordering
- Be cryptographically verifiable
- Be reconstructable across distributed systems
- Serve as the canonical audit trail for all SDLP assets

Logs MUST NOT be modified, deleted, or reordered.

48.4 Log Entry Structure

Each log entry MUST be represented as an S3F object containing:

LogID
: Globally unique identifier.

EventID
: Identifier of the associated event.

Timestamp
: RFC 3339 timestamp of log creation.

DistributorID
: Distributor responsible for the event.

AssetID (if applicable)
: Asset associated with the event.

EventType
: Canonical event type from the SDLP Event Type Registry.

EventPayload
: S3F object containing event-specific data.

PreviousLogHash
: Hash of the previous log entry (chain linking).

LogHash
: Hash of the current log entry.

48.5 Log Immutability Requirements

SDLP logs MUST be immutable. The following rules apply:

- No log entry may be altered after creation.
- No log entry may be deleted.
- No log entry may be inserted retroactively.
- Corrections MUST be recorded as new log entries.
- LogHash and PreviousLogHash MUST form a verifiable chain.

Any violation MUST trigger SafeMode and a LoggingIntegrityFailure event.

48.6 Log Ordering Rules

Logs MUST be strictly ordered by Timestamp. Systems MUST:

- Reject logs with timestamps earlier than the previous entry.
- Reject logs with identical timestamps unless sequence numbers are used.
- Ensure deterministic ordering across distributed systems.

48.7 Log Retention Requirements

Logs MUST be retained:

- For the lifetime of the asset
- For at least 10 years after Bitdump
- Indefinitely for governance-critical events

Logs MUST remain accessible for audit and compliance operations.

48.8 Log Synchronization Rules

SDLP-compliant systems MUST support cross-system synchronization:

- Logs MUST be replicated across trusted registries.
- Synchronization MUST be deterministic and conflict-free.
- Conflicts MUST be resolved using timestamp and LogHash ordering.
- Missing logs MUST be reconstructed from peer systems.

Synchronization events MUST be logged as LogSyncPerformed.

48.9 Logging Failure Detection

Systems MUST detect:

- Missing logs
- Corrupted logs
- Out-of-order logs
- Hash chain breaks
- Duplicate LogIDs
- Invalid EventPayload structures

Detection MUST trigger LoggingIntegrityFailure.

48.10 Logging Failure Handling

If logging fails:

- The asset MUST enter SafeMode.
- A LoggingIntegrityFailure event MUST be logged.
- The distributor MUST be notified immediately.
- No lifecycle events MAY proceed until remediation.

48.11 Log Reconstruction Rules

Reconstruction MUST:

- Use surviving logs from distributed registries
- Validate all LogHash and PreviousLogHash links
- Rebuild missing entries using deterministic replay
- Reject unverifiable or conflicting entries

Reconstruction MUST be logged as LogReconstructed.

48.12 Governance Logging Requirements

Governance systems MUST log:

- Policy updates
- Policy conflicts
- Policy enforcement failures
- Threshold changes
- Governance reviews
- Distributor sanctions or reinstatements

Governance logs MUST be retained indefinitely.

48.13 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system log verification
- Deterministic log replay
- Forward compatibility with new log fields
- Backward compatibility with legacy log formats

49. SDLP-AUDIT ARCHITECTURE AND CROSS-SYSTEM VERIFICATION FRAMEWORK

49.1 Purpose

This section defines the SDLP Audit Architecture (SAA), including the rules, procedures, verification models, and cross-system requirements for auditing SDLP assets, logs, distributors, and lifecycle events. Audits ensure that SDLP-compliant systems remain trustworthy, verifiable, and resistant to corruption or unauthorized modification.

49.2 Scope

These requirements apply to:

- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance and Governance Systems

49.3 Audit Model Overview

SDLP audits MUST:

- Validate asset integrity
- Validate lifecycle correctness
- Validate compliance with SDLP policies
- Validate log immutability and completeness
- Validate distributor authentication and authorization history
- Validate cross-system consistency

Audits MUST be deterministic, reproducible, and cryptographically verifiable.

49.4 Audit Types

SDLP supports the following audit types:

IntegrityAudit
: Verifies hashes, signatures, and metadata correctness.

ComplianceAudit
: Verifies adherence to SDLP policies and thresholds.

LifecycleAudit
: Verifies lifecycle transitions, ordering, and validity.

LogAudit
: Verifies log chain integrity and completeness.

DistributorAudit
: Verifies distributor identity, credentials, and session history.

GovernanceAudit
: Verifies policy enforcement, conflicts, and overrides.

CrossSystemAudit
: Verifies consistency across distributed SDLP registries.

49.5 Audit Trigger Conditions

Audits MUST be triggered by:

- Scheduled audit intervals
- Compliance failures
- Integrity failures
- Policy conflicts
- SafeMode entry or exit
- Distributor credential changes
- Cross-system synchronization events

- Governance review requirements

49.6 Audit Record Structure

Each audit MUST produce an S3F AuditRecord containing:
AuditID
AuditType
Timestamp
DistributorID (if applicable)
AssetID (if applicable)
AuditScope
Findings
Violations
RemediationRequired
AuditSignature

Audit records MUST be logged as AuditPerformed events.

49.7 Audit Verification Requirements

Audits MUST verify:
- All hashes match expected values
- All lifecycle transitions follow canonical rules
- All logs form an unbroken hash chain
- All policies were enforced correctly
- All thresholds were respected
- All decay, entropy, ASI, and P-value indices are consistent
- All lineage references are valid
- All distributor sessions were authenticated and authorized

49.8 Cross-System Audit Requirements

Cross-system audits MUST:
- Compare logs across distributed registries
- Detect missing, conflicting, or divergent entries
- Validate hash chain consistency across systems
- Validate synchronized policy versions
- Validate synchronized lifecycle states
- Reconstruct missing logs when possible

Cross-system discrepancies MUST be logged as CrossSystemAuditFailure.

49.9 Audit Failure Handling

If an audit fails:
- The asset MUST enter Restricted or SafeMode
- A ComplianceReviewRequired event MUST be logged
- The distributor MUST be notified immediately
- Governance systems MUST be alerted
- No lifecycle events MAY proceed until remediation

49.10 Remediation Requirements

Remediation MAY include:
- Rebuilding missing logs
- Recomputing hashes
- Correcting metadata
- Reconstructing lineage
- Revalidating distributor credentials
- Reapplying policy thresholds
- Recalculating decay, entropy, ASI, and P-values

Remediation MUST be validated by a follow-up audit.

49.11 Audit Chain of Custody

All audit artifacts MUST:
- Be cryptographically signed
- Be retained indefinitely for governance audits
- Be immutable once finalized
- Be reproducible from logs and metadata

49.12 Governance Oversight

Governance systems MUST:
- Review repeated audit failures
- Enforce distributor sanctions when necessary
- Update policies in response to systemic issues
- Maintain the global audit registry

49.13 Interoperability Requirements

All SDLP-compliant systems MUST support:
- Cross-system audit replay
- Deterministic audit verification
- Forward compatibility with new audit types
- Backward compatibility with legacy audit formats

50. SDLP-DISTRIBUTOR ARCHITECTURE AND AUTHORIZED OPERATION MODEL

50.1 Purpose

This section defines the SDLP Distributor Architecture (SDA), including identity requirements, authentication rules, authorization constraints, operational boundaries, and compliance obligations for all SDLP distributors. Distributors are the primary trust anchors in the SDLP ecosystem and MUST adhere to strict operational standards.

50.2 Scope

These requirements apply to:
- All SDLP Distributors
- All SDLP Asset Registries
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Governance Systems

50.3 Distributor Identity Requirements

Each distributor MUST be assigned a globally unique DistributorID.
Distributor identity MUST include:
DistributorID
DistributorName
DistributorPublicKey
DistributorPolicySet
DistributorCapabilities
DistributorStatus (Active, Suspended, Revoked)

Distributor identity MUST be stored in the SDLP Global Registry.

50.4 Distributor Authentication Requirements

Distributors MUST authenticate using:
- Public-key cryptography
- Non-replayable session tokens
- Time-bound authentication windows
- Mutual TLS or equivalent secure transport

Authentication MUST be logged as DistributorAuthentication.

50.5 Distributor Authorization Model

After authentication, distributors MUST be authorized for specific operations. Authorization MUST be based on:

- DistributorCapabilities
- Policy enforcement rules
- Asset lifecycle state
- Session context
- Governance restrictions

Unauthorized operations MUST be rejected and logged as DistributorAuthorizationFailure.

50.6 Distributor Operational Capabilities

Distributors MAY be authorized to perform the following operations:

- Asset issuance
- Asset transfer
- Asset duplication
- Asset transformation
- Metadata updates
- Lifecycle transitions
- Compliance checks
- Integrity checks
- Policy evaluation
- Log synchronization
- Audit initiation

Capabilities MUST be explicitly declared and MUST NOT be inferred.

50.7 Distributor Session Rules

Distributor sessions MUST:

- Be time-limited
- Be cryptographically bound to the distributor identity
- Include a unique SessionID
- Be logged at creation and termination
- Be invalidated upon credential revocation

Session termination MUST be logged as SessionClosed.

50.8 Distributor Compliance Obligations

Distributors MUST:

- Enforce all SDLP policies
- Maintain accurate logs
- Support audits
- Maintain secure infrastructure
- Protect private keys
- Maintain policy version synchronization
- Support SafeMode and quarantine operations

Failure to meet obligations MUST trigger GovernanceReviewRequired.

50.9 Distributor Suspension Rules

A distributor MUST be suspended if:

- Multiple policy violations occur
- Multiple audit failures occur
- Credential compromise is detected
- Logging integrity is compromised
- Governance systems mandate suspension

Suspension MUST be logged as DistributorSuspended.

50.10 Distributor Revocation Rules

A distributor MUST be revoked if:

- Malicious activity is confirmed
- Repeated suspension events occur
- Cryptographic keys are irrecoverably compromised
- Governance systems determine the distributor is untrustworthy

Revocation MUST be logged as DistributorRevoked.

50.11 Distributor Recovery Rules

A suspended distributor MAY be reinstated if:
- All audit failures are remediated
- All policy violations are resolved
- New cryptographic credentials are issued
- Governance systems approve reinstatement

Reinstatement MUST be logged as DistributorReinstated.

50.12 Distributor Event Requirements

Distributors MUST generate the following events:
DistributorAuthentication
DistributorAuthorizationFailure
DistributorSuspended
DistributorRevoked
DistributorReinstated
SessionClosed

All events MUST follow S3F and SETR requirements.

50.13 Interoperability Requirements

All SDLP-compliant systems MUST support:
- Cross-system distributor verification
- Deterministic distributor capability evaluation
- Forward compatibility with new distributor classes
- Backward compatibility with legacy distributor models

51. TERMINAL STATE SEMANTICS AND LIFECYCLE FINALITY

51.1 Purpose

This section defines the architectural meaning of Terminal States within the SDLP lifecycle model. Terminal States represent lifecycle endpoints for a specific SDLP instance, beyond which no additional transitions, modifications, or continuations of that instance are permitted.

51.2 Definition of Terminal States

A Terminal State is any lifecycle state designated as irreversible within the SDLP lifecycle taxonomy. Examples include, but are not limited to:
- Expired
- Revoked
- Superseded
- Bitdumped
- Zeroized

These states represent the conclusion of the lifecycle of a particular SDLP instance.

51.3 Instance-Level Finality

Once an SDLP instance enters a Terminal State:
- No further lifecycle events may be appended to that instance.
- That instance may not be resumed, reactivated, or continued

under any lifecycle state.
- That instance ceases to participate in distribution, transfer, or update workflows.

Terminality applies to the instance itself, not to the underlying work or to the customer relationship.

51.4 New Issuance Versus Continuation

51.4.1 New Issuance Permitted by Policy

SDLP does not prohibit a Distributor from issuing a new SDLP instance of the same underlying digital work to a legitimate purchaser, provided such issuance is explicitly allowed by EULA, license terms, or applicable policy.

51.4.2 Eligibility Constraint: Customer Status

A new instance may be issued only if the Customer is not currently flagged for unauthorized use. A Customer is considered ineligible if multiple Lifecycle Discrepancy Reports (LDRs) associate unauthorized use with the Customers identity attributes, or if any active restriction is in effect.

51.4.3 Distinct Identity Requirement

Any new issuance:

- MUST use a new SDLP ID.
- MUST be treated as a separate instance.
- MUST maintain an independent lifecycle and provenance chain.
- MUST NOT imply continuation or reactivation of any prior instance.

51.4.4 No Implicit Reinstatement

Issuing a new instance MUST NOT alter, reopen, or reinterpret the Terminal State of any previous instance. Terminality applies per instance.

51.5 Prohibited Transitions

For a given SDLP instance, the following transitions are architecturally invalid:

- Terminal ? Active
- Terminal ? Transitional
- Terminal ? Derivative (of itself)
- Terminal ? Terminal (re-terminalization)

Any attempt to perform such transitions constitutes a lifecycle model violation.

51.6 Rationale

Terminal States ensure lifecycle determinism, provenance integrity, and predictable behavior across distributed SDLP implementations. This model preserves per-instance finality while allowing legitimate reissuance when permitted by EULA and when the Customer is not flagged for unauthorized use.

52. ZEROIZATION SEMANTICS AND IRREVERSIBLE DESTRUCTION STATES

52.1 Purpose

This section defines the architectural meaning of Zeroization and other irreversible destruction states within the SDLP lifecycle model. These states represent the conceptual end of an SDLP instances existence, after which no further lifecycle participation is possible.

52.2 Definition of Zeroization

Zeroization is the architectural designation for the irreversible destruction of an SDLP instances functional identity, state, and lifecycle continuity. Zeroization marks the point at which an instance ceases to exist as a coherent lifecycle participant.

52.3 Bitdump as Canonical Destruction State

Bitdump is the canonical Zeroization-class state within the SDLP lifecycle taxonomy. When an instance enters the Bitdumped state, its lifecycle is concluded and its identity is considered permanently terminated.

Bitdump is treated as the definitive endpoint for destruction semantics, though other destruction-class states may exist within the taxonomy.

52.3.1 Destruction Intent

Bitdump represents the architectural principle that an SDLP instance must prefer irreversible termination over continued existence in a corrupted, compromised, or unauthorized state.

52.3.2 Destruction Analogy

Zeroization-class states follow the same conceptual model as tamper-reactive physical loss-prevention devices. If an SDLP instance is subjected to unauthorized manipulation, the lifecycle model requires the instance to prefer irreversible destruction over continued existence in a compromised state.

52.4 Instance-Level Destruction Finality

Once an SDLP instance enters a Zeroization-class state:

- The instances lifecycle is permanently closed.
- No additional lifecycle events may be appended.
- The instance may not transition to any active or transitional state.
- The instance ceases to participate in distribution, transfer, update, or lineage workflows.

Destruction finality applies per instance and does not affect the Distributors ability to issue new instances under applicable policy.

52.5 Mandatory Self-Reporting

Upon entering a Zeroization-class state, an SDLP instance SHALL emit a destruction report documenting the terminal transition. This report provides verifiable evidence of the lifecycle violation or termination event and ensures that the broader ecosystem is aware of the instances irreversible cessation.

52.6 Prohibited Post-Destruction Behaviors

For a Zeroized or Bitdumped instance, the following behaviors are architecturally invalid:

- Accepting new lineage entries
- Accepting new policies or configuration
- Validating environment trust signals
- Performing lifecycle operations
- Re-entering any non-terminal state

52.7 Relationship to Terminal States

Zeroization-class states are a subset of Terminal States. While all Zeroization-class states are Terminal, not all Terminal States imply

destruction. Zeroization specifically denotes irreversible loss of functional identity and lifecycle continuity.

52.8 Rationale

Zeroization semantics ensure that SDLP-governed objects have a clearly defined and irreversible endpoint within the lifecycle model. These semantics provide architectural clarity for destruction behavior while allowing enforcement mechanisms to be defined separately in the SDLP Security Architecture.

53. REHYDRATION SEMANTICS, RESURRECTION PROHIBITION, AND ARCHITECTURAL ANTI-REINSTANTIATION PRINCIPLES

53.1 Architectural Finality of Destruction

Once an SDLP instance enters a Zeroization-class state, its lifecycle is permanently concluded. The instance ceases to exist as a lifecycle participant and MUST NOT be treated as recoverable, transitional, or capable of further evolution. Destruction finality is a foundational component of SDLP lifecycle physics.

53.2 Rehydration Semantics

Rehydration refers to any attempt to reconstruct, reload, restore, or reassemble an SDLP instance after destruction. All such attempts are architecturally invalid and MUST be rejected by any SDLP-compliant system.

53.3 Resurrection Prohibition

Resurrection refers to any attempt to revive a destroyed instance by reintroducing it into the lifecycle model. Resurrection is strictly prohibited and constitutes a lifecycle violation.

53.4 Anti-Reinstantiation Principles

Reinstantiation refers to creating a new instance that claims continuity with a destroyed instance. This is architecturally invalid. Any new issuance MUST be treated as a distinct instance with no continuity to the destroyed one.

53.5 Rationale

These principles ensure lifecycle determinism, prevent identity corruption, and maintain the integrity of Zeroization-class states.

54. INITIALIZATION PHYSICS AND PRE-INIT TERMINATION SEMANTICS

54.1 Purpose

This section defines the architectural meaning of Initialization within the SDLP lifecycle model. Initialization is the transition point at which an encoded, inert SDLP instance evaluates its host environment to determine whether lifecycle activation may safely begin. Initialization is a mandatory trust boundary that prevents unauthorized activation, corruption, or misuse.

54.2 Initialization as a Lifecycle Boundary

Initialization represents the boundary between inert encoding and active lifecycle participation. Prior to Initialization, the instance is non-functional and incapable of participating in any lifecycle operations. Initialization MUST occur before any lifecycle state may be entered.

54.3 Pre-Init Termination

If the host environment fails trust evaluation, the instance MUST terminate prior to activation. Pre-init termination is mandatory when:

- Environment trust cannot be established
- Required metadata is missing or corrupted
- Policy thresholds are violated
- Distributor authentication cannot be validated
- Physics consistency checks fail (decay, entropy, ASI, P-values)

Pre-init termination MUST NOT allow partial activation or partial lifecycle entry.

54.4 Initialization Requirements

Initialization MUST verify:

- Environment trust signals
- Distributor identity and cryptographic keys
- Policy version alignment
- Metadata completeness and correctness
- Hash and signature validity
- Physics consistency (decay, entropy, ASI, P-values)
- Session context validity

Initialization MUST be deterministic and reproducible.

54.5 Initialization Failure Handling

If Initialization fails:

- The instance MUST NOT activate
- A PreInitTermination event MUST be logged
- No lifecycle state may be entered
- The distributor MUST be notified

Initialization failure MUST NOT allow remediation unless explicitly permitted by policy.

54.6 Catastrophic Initialization Failure

If Initialization detects tampering, corruption, or malicious environmental indicators:

- The instance MUST undergo immediate Bit-Drop
- A BitDropInitiated event MUST be logged
- A BitDropCompleted event MUST be logged

Catastrophic failures MUST NOT allow SafeMode or Restricted state.

54.7 Initialization Success Conditions

Initialization MAY proceed only when:

- All trust signals validate
- All metadata is complete and correct
- All physics indices are within acceptable thresholds
- All policies are aligned and active
- Distributor authentication is valid
- No tampering indicators are present

Successful Initialization MUST be logged as InitializationCompleted.

54.8 Rationale

Initialization ensures that SDLP instances activate only in trusted, policy-compliant environments. This boundary prevents unauthorized activation, ensures lifecycle correctness from the first moment of existence, and provides the earliest possible enforcement point for Bit-Drop in hostile or compromised environments.

55. SDLP-TRUST MODEL AND ENVIRONMENTAL ATTESTATION FRAMEWORK

55.1 Purpose

This section defines the SDLP Trust Model and the Environmental Attestation Framework (EAF). These mechanisms determine whether a host environment is sufficiently trustworthy for an SDLP instance to initialize, operate, or continue its lifecycle. Trust evaluation is a mandatory prerequisite for Initialization, SafeMode exit, and prevention of unauthorized manipulation.

55.2 Scope

These requirements apply to:

- All SDLP instances
- All SDLP Distributors
- All SDLP Lifecycle Event Handlers
- All SDLP Verification Engines
- All SDLP Compliance and Governance Systems

55.3 Trust Model Overview

The SDLP Trust Model defines the rules by which an instance evaluates its execution environment. Trust MUST be:

- Deterministic
- Cryptographically verifiable
- Non-spoofable
- Non-bypassable
- Logged and auditable

Trust MUST NOT rely on environmental assumptions, heuristics, or unverifiable signals.

55.4 Environmental Attestation Requirements

Prior to Initialization or lifecycle continuation, an SDLP instance MUST perform Environmental Attestation. Attestation MUST verify:

- Distributor authentication
- Policy version alignment
- Integrity of the execution environment
- Presence of required cryptographic materials
- Absence of tampering indicators
- Validity of session context
- Compliance with environmental policy thresholds

Attestation MUST be performed using deterministic, reproducible methods.

55.5 Attestation Signals

The following signals MUST be evaluated:

TrustAnchorStatus
: Verification of distributor identity and cryptographic keys.

PolicySetIntegrity

Verification that active policies match expected versions.

EnvironmentIntegrity
: Verification of host environment integrity, including anti-tamper indicators.

SessionValidity
: Verification of session tokens, expiration, and replay protection.

MetadataCompleteness
: Verification that required metadata fields are present and valid.

PhysicsConsistency
: Verification that decay, entropy, ASI, and P-values are consistent with expected lifecycle physics.

55.6 Attestation Outcomes

Attestation MUST result in one of the following outcomes:

Trusted
: Environment is valid and compliant. Initialization or lifecycle continuation MAY proceed.

Untrusted
: Environment fails one or more attestation checks. The instance MUST NOT initialize or continue.

Catastrophic
: Environment exhibits tampering, corruption, or malicious indicators. The instance MUST undergo immediate Bit-Drop.

55.7 Mandatory Actions for Untrusted Environments

If attestation results in an Untrusted outcome:

- Initialization MUST NOT proceed.
- The instance MUST remain inert.
- A PreInitTermination event MUST be logged.
- The distributor MUST be notified.

No lifecycle state may be entered until trust is re-established.

55.8 Mandatory Actions for Catastrophic Environments

If attestation results in a Catastrophic outcome:

- The instance MUST undergo immediate Bit-Drop.
- A BitDropInitiated event MUST be logged.
- A BitDropCompleted event MUST be logged.
- The environment MUST be flagged for GovernanceReviewRequired.

Catastrophic outcomes MUST NOT allow SafeMode or remediation.

55.9 Continuous Trust Evaluation

Trust MUST be continuously evaluated during:

- Lifecycle transitions
- Policy updates
- Distributor authentication
- SafeMode exit
- Audit operations
- Log synchronization

Any trust failure MUST trigger Restricted state or SafeMode.

55.10 Trust Decay Model

Trust MAY degrade over time or due to environmental instability. Systems MUST:

- Track TrustDecayIndex
- Trigger Restricted state when TrustDecayIndex exceeds TrustWarningThreshold
- Trigger SafeMode when TrustDecayIndex exceeds TrustCriticalThreshold

Trust decay MUST be monotonic unless remediated.

55.11 Trust Remediation Requirements

Trust MAY be restored through:

- Reauthentication
- Policy synchronization
- Environment integrity repair
- Session regeneration
- Metadata correction

Remediation MUST be validated before trust is restored.

55.12 Logging Requirements

All trust evaluations MUST generate:

TrustEvaluated
- Outcome
- Signals evaluated
- Timestamp
- DistributorID

TrustFailure
- Reason
- Failed signals
- Timestamp

TrustRestored
- Remediation actions
- Validation summary

55.13 Interoperability Requirements

All SDLP-compliant systems MUST support:

- Cross-system trust verification
- Deterministic attestation replay
- Forward compatibility with new trust signals
- Backward compatibility with legacy trust models

56. IANA Considerations

This document has no IANA actions.

57. Author’s Address

M. Norton
Email: mark433norton@gmail.com