Network Working Group
Internet-Draft
Intended status: Informational
Expires: 24 August 2026

J. Tejido

20 February 2026

SlimWire Protocol (SWP) Core: Binary Framing and Envelope Substrate
draft-tejido-swp-core-00

Abstract

This document specifies the SlimWire Protocol (SWP) Core: a slim binary framing (record) layer and envelope format for interoperable agent/tool messaging with profile-based extensibility and a machine-verifiable conformance model. SWP Core provides a transport-independent message boundary definition (u32_be length prefix), a mandatory-to-implement envelope encoding (E1) based on unsigned LEB128 uvarints and length-delimited byte strings with a TLV extension block, and profile dispatch by numeric profile identifiers. Profiles define application semantics (e.g., tool invocation mapping and agent task lifecycle) and are carried opaquely by Core. This document also specifies conformance classes and golden vector artefacts executed by a spec-vector runner, and defines a security binding (S1) requiring an authenticated confidential channel (e.g., TLS 1.3).

Status of This Memo

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

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

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

This Internet-Draft will expire on 24 August 2026.

Copyright Notice

Table of Contents

1. Introduction

Contemporary LLM systems increasingly behave as distributed systems: they invoke external tools, delegate long-running tasks to other agents, stream intermediate progress, exchange artefacts, and operate across organisational trust boundaries. In practice, these behaviours are implemented via heterogeneous application protocols over transports such as HTTP streaming and stdio.

Two application-level protocol efforts have gained traction: the Model Context Protocol (MCP) (https://modelcontextprotocol.io/specification/ (https://modelcontextprotocol.io/specification/)) and Agent2Agent (A2A) (https://a2a-protocol.org/ (https://a2a-protocol.org/); overview: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/ (https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/)). MCP standardises tool invocation and discovery (commonly JSON-RPC over defined transports), while A2A standardises agent-to-agent delegation and discovery. Both operate at the application layer.

As deployments move into federated, cross-boundary settings, integration friction often arises below the application layer: message boundary detection, size limiting, replay policy, correlation, identity surfacing for authorisation and audit, and failure semantics are repeatedly reimplemented at gateways and adapters. SWP addresses this by defining a minimal substrate: a record layer and envelope that carry multiple profiles under a stable dispatch model, with explicit security binding requirements and a machine-verifiable conformance suite.

This document specifies SWP Core (framing, envelope, E1 encoding, and profile dispatch), plus the conformance and security binding requirements needed for interoperable deployments. SWP profiles (including mappings for MCP and A2A and additional infrastructure primitives) are referenced but are not fully specified herein unless required for Core interoperability.

2. Conventions and Terminology

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

BCP 14 is specified by RFC 2119 (https://www.rfc-editor.org/rfc/rfc2119 (https://www.rfc-editor.org/rfc/rfc2119)) and clarified by RFC 8174 (https://www.rfc-editor.org/rfc/rfc8174 (https://www.rfc-editor.org/rfc/rfc8174)).

This document uses the following terms:

* _Frame_: A length-delimited unit on a byte stream consisting of a 32-bit big-endian length prefix and an encoded envelope.

* _Envelope_: The Core-level header containing versioning, profile dispatch identifiers, correlation identifiers, and an opaque profile payload.

* _Profile_: A defined semantic layer above Core identified by a numeric _profile_id_ and a profile-defined _msg_type_. Profiles define payload schemas, processing rules, and conformance vectors.

* _Binding_: A set of requirements that constrain how Core and profiles are carried over a transport (e.g., security binding S1; payload encoding binding P1).

3. SWP Core Overview

SWP Core defines (1) message framing on a byte stream, (2) a minimal envelope surface for dispatch, and (3) a mandatory envelope encoding (E1). Profiles define application semantics and payload formats and are carried as opaque bytes in the Core envelope.

3.1. Core Framing

A SWP stream is a sequence of frames. Each frame MUST be encoded as:

* _N_: a 32-bit unsigned integer in network byte order (big-endian), followed by

* exactly _N_ octets containing one encoded envelope.

A receiver MUST reject a frame if any of the following conditions hold:

* the 4-octet length prefix is truncated;

* _N_ equals zero;

* _N_ exceeds a local maximum frame size limit (MAX_FRAME_BYTES);

* the envelope body is truncated (fewer than _N_ octets remain);

* the envelope body cannot be decoded by the selected envelope encoding (E1 for Core v1).

MAX_FRAME_BYTES is a local policy parameter and MUST be configurable. Implementations SHOULD enforce MAX_FRAME_BYTES before allocating buffers sized by _N_.

NOTE: SWP Core is transport-independent and can run over any reliable byte stream meeting the security binding (S1). HTTP/2 provides a distinct binary framing layer for HTTP semantics (RFC 9113, https://datatracker.ietf.org/doc/html/rfc9113 (https://datatracker.ietf.org/doc/html/rfc9113)); gRPC defines an additional message framing convention over HTTP/2 (https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md (https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md)). SWP similarly defines explicit message boundaries but is not tied to HTTP/2.

3.2. Core Envelope Fields

A decoded envelope consists of the following required fields:

* _version_ (uvarint): Core protocol version (this document specifies version 1).

* _profile_id_ (uvarint): profile dispatch identifier.

* _msg_type_ (uvarint): profile-defined message type.

* _flags_ (uvarint): Core behaviour flags. Unknown flag bits MUST NOT change interpretation.

* _ts_unix_ms_ (uvarint): sender timestamp in Unix milliseconds (policy-controlled freshness).

* _msg_id_ (bytes): message identifier for correlation and replay/duplication handling.

* _payload_ (bytes): opaque profile payload bytes.

The Core MUST NOT interpret profile payload bytes. Dispatch decisions MUST be based only on (version, profile_id, msg_type, flags) and local policy/state.

msg_id length MUST be bounded. Implementations MUST reject msg_id values shorter than MIN_MSG_ID_BYTES or longer than MAX_MSG_ID_BYTES. MIN_MSG_ID_BYTES and MAX_MSG_ID_BYTES are protocol parameters; this document RECOMMENDS MIN_MSG_ID_BYTES = 8 and MAX_MSG_ID_BYTES = 64.

The Core MUST enforce a maximum payload size limit (MAX_PAYLOAD_BYTES) independent of MAX_FRAME_BYTES. MAX_PAYLOAD_BYTES MUST be configurable and SHOULD be strictly less than MAX_FRAME_BYTES to allow envelope overhead.

4. E1 Envelope Encoding

E1 is the mandatory-to-implement envelope encoding for Core version 1. E1 encodes envelope fields in a fixed order using unsigned LEB128 uvarints and length-delimited byte strings. E1 includes a TLV extension block that can be skipped by implementations that do not recognise extension types.

4.1. Encoding Primitives

E1 defines two primitive encodings:

* _uvarint_: an unsigned integer encoded using base-128 LEB128, where each octet carries 7 bits of value and the most significant bit indicates continuation. The least significant group is encoded first.

* _bytes_: a length-delimited octet string encoded as _uvarint length_ followed by exactly _length_ octets.

An E1 decoder MUST reject malformed varints and MUST enforce a varint length bound. This document RECOMMENDS rejecting uvarints longer than 10 octets and rejecting values that overflow 64-bit unsigned range.

4.2. Field Order and Layout

Within the frame body (the _N_ octets after the length prefix), E1 encodes the envelope fields in the following exact order:

1. _version_ (uvarint)

2. _profile_id_ (uvarint)

3. _msg_type_ (uvarint)

4. _flags_ (uvarint)

5. _ts_unix_ms_ (uvarint)

6. _msg_id_ (bytes)

7. _extensions_ (bytes; TLV extension block; MAY be empty)

8. _payload_ (bytes)

A decoder MUST treat absence of any required field as an invalid envelope. A decoder MUST NOT reinterpret unknown flag bits or unknown extension types. Unknown extension types MUST be ignored.

4.3. TLV Extension Block

The _extensions_ field is itself a length-delimited byte string. Its content is a sequence of TLV entries. Each entry is encoded as:

* _ext_type_ (uvarint)

* _ext_value_ (bytes)

This construction is TLV because _ext_value_ is length-delimited. Unknown ext_type values MUST be ignored and skipped by reading ext_value length and advancing that many octets.

Implementations MUST enforce an upper bound on extension block size (MAX_EXT_BYTES) and MUST reject messages whose extension block length exceeds that bound. MAX_EXT_BYTES is a local policy parameter and SHOULD be configurable.

Extension type allocation is governed by the profile registry policy. Low ranges are reserved for Core/bindings and higher ranges for profile-defined extensions. This document does not define a global extension registry beyond these allocation principles.

5. P1 Payload Encoding Binding

SWP Core treats payload bytes opaquely. To enable independent implementability and deterministic conformance testing for profile payloads, SWP defines a mandatory payload encoding binding P1 for profiles that opt into structured payload encoding.

P1 specifies Protocol Buffers (proto3) wire format as mandatory-to-implement for applicable profiles. Protobuf proto3 guidance is documented at https://protobuf.dev/programming-guides/proto3/ (https://protobuf.dev/programming-guides/proto3/).

Rationale (informative): Protobuf supports compact binary payloads, established forward/backward compatibility discipline (stable field numbers; unknown fields ignored), and byte-level golden vectors.

Normative schema annexes: this specification set includes normative .proto files for P1 profile payloads (e.g., SWP RPC, events, artifacts, and A2A-mapped messages). The .proto annexes are

distributed with the SWP spec kit and are referenced by profile specifications. Implementations claiming conformance to a profile that uses P1 MUST decode payloads using the corresponding normative .proto schema.

Exception: the MCP mapping profile carries MCP JSON-RPC payload bytes opaquely to preserve MCP semantics and avoid semantic drift. MCP specification is available at https://modelcontextprotocol.io/specification/ (https://modelcontextprotocol.io/specification/).

6. Profile Registry and profile_id Allocation

Core dispatch depends on a stable registry of numeric profile identifiers (profile_id). profile_id values MUST NOT be reused. A profile specification MUST define its profile_id, msg_type space, payload binding (e.g., P1 protobuf or opaque), and conformance vectors.

This document defines the following initial allocation guidance:

* profile_id 1: MCP mapping profile (opaque JSON-RPC bytes).
* profile_id 2: A2A profile (agent task lifecycle and related messages).
* profile_id 3-9: reserved for future foundational profiles/bindings.
* profile_id 10-19: reserved for extended infrastructure primitive profiles.

Allocation policy (informative): new profile_id assignments SHOULD be accompanied by (1) a profile specification, (2) conformance vectors for core behaviours and error paths, and (3) an explicit compatibility statement. Experimental/private-use identifiers should be confined to a clearly segregated range as defined by the profile registry document for the SWP spec kit.

7. Conformance Classes and Golden Vectors

SWP defines conformance classes (C0 through C5) to enable scoped implementation claims. Each class corresponds to a set of required vector namespaces. Implementations MUST state which conformance class claims are supported.

7.1. Conformance Classes

Conformance classes are cumulative. A higher class includes all requirements of lower classes. The class taxonomy is:

Class	Intended claim	Required components (illustrative)
C0	Core baseline	Core framing + E1 encoding + S1 binding requirements
C1	MCP bridge	C0 + MCP mapping profile
C2	A2A baseline	C0 + A2A profile
C3	Runtime primitives	C0 + RPC + EVENTS + OBS profiles
C4	Data plane	C3 + ARTIFACT + STATE profiles
C5	Federation	C4 + discovery + tool discovery + credential + policy hint + relay profiles

Table 1: SWP Conformance Classes

7.2. Golden Vector Format

The conformance suite consists of golden vectors. Each vector pairs:

* a binary fixture (.bin) containing the exact on-the-wire frame bytes (u32_be length + E1 envelope bytes);

* a JSON descriptor (.json) containing expected outcome (accept/reject), expected error codes, and assertions.

Vectors include positive cases, negative cases exercising each rejection path, and boundary cases for size limits and reserved/unknown behaviours. Profiles MUST provide vectors for required invariants and error handling.

A spec-vector runner executes vectors and emits a reproducible JSON summary. The runner supports a strict "no-fallback" mode that fails any vector requiring fallback evaluation. This enables a clear distinction between (a) specification-level validation and (b) implementation-only conformance runs.

The SWP spec kit includes runner output schema documentation and artefact bundle guidance. Implementations SHOULD publish artefact bundles (stdout logs and JSON summaries) when making public conformance claims.

Wire fixture (.bin):
[u32_be N] [E1 envelope bytes]

E1 layout (field order):
version(uvarint)=1
profile_id(uvarint)=1
msg_type(uvarint)=1
flags(uvarint)=0
ts_unix_ms(uvarint)=0
msg_id(bytes)=len 16 + 16 octets
extensions(bytes)=len 0
payload(bytes)=len 0

Expected (.json):

```
{
  "vector_id": "e1_0001_valid_min_envelope",
  "expected": {
    "outcome": "accept",
    "assert": {
      "version": 1,
      "profile_id": 1,
      "msg_type": 1,
      "payload_len": 0
    }
  }
}
```

Figure 1: Example Conformance Vector (Illustrative)

8. Security Considerations

SWP Core is intended for federated deployments and therefore specifies a security binding (S1) that constrains channel properties required before frames are processed.

8.1. S1 Security Binding

S1 requires that SWP frames be carried over an authenticated confidential channel providing: confidentiality, integrity, peer authentication, replay resistance, and downgrade resistance.

A concrete and widely deployed instantiation is TLS 1.3 (RFC 8446, https://datatracker.ietf.org/doc/html/rfc8446 (https://datatracker.ietf.org/doc/html/rfc8446)). When TLS is used, implementations SHOULD use deployment-appropriate authentication (e.g., mutual TLS) and MUST fail closed on authentication or integrity failures.

SWP Core includes a sender timestamp (ts_unix_ms) that MAY be used for freshness enforcement. If freshness enforcement is enabled, receivers MUST reject frames outside a configured acceptance window. If freshness enforcement is disabled, this MUST be documented by the implementation and deployment.

8.2. Credential/Delegation and Policy Hints

Federated agent/tool systems often require delegation ("act on behalf of") and constraint propagation. SWP profiles may carry credential and delegation artefacts as opaque tokens with explicit expiry and chain-length limits, and may carry policy hints (e.g., residency restrictions, cost budgets, network restrictions) with defined conflict and violation signalling.

These profiles are intentionally minimal and do not define a full IAM system or policy language. They provide a portable signalling surface suitable for relays and cross-organisation governance.

8.3. Observability Context Propagation

Distributed tracing propagation is standardised by W3C Trace Context (https://www.w3.org/TR/trace-context/ (https://www.w3.org/TR/trace-context/)). OpenTelemetry adopts W3C Trace Context as the default propagation format and documents context propagation behaviour at https://opentelemetry.io/docs/concepts/context-propagation/ (https://opentelemetry.io/docs/concepts/context-propagation/). SWP observability profiles align with these standards to enable cross-hop trace correlation without proprietary propagation conventions.

Implementations SHOULD avoid propagating sensitive identifiers across organisational boundaries and SHOULD apply data minimisation to propagated context.

9. Implementation Status

The SWP spec kit is accompanied by an executable proof-of-concept (PoC) implementation used to generate and validate conformance vectors via the spec-vector runner. The PoC is a reference artefact rather than a production deployment, intended to exercise framing, envelope decoding/encoding, profile dispatch, and conformance evaluation end-to-end.

Earlier iterations used JSON payload parsing for some profile handlers while specifying P1 protobuf payload binding as normative. This mismatch has been resolved: P1 protobuf payload binding is now both normative for applicable profiles and implemented in the PoC for spec-vector execution. MCP mapping remains byte-preserving and carries JSON-RPC bytes opaquely to avoid semantic drift.

10. Evaluation and Reproducibility

SWP evaluation is grounded in machine-verifiable conformance artefacts. The spec-vector runner executes golden vectors and emits a JSON summary with a versioned schema, run provenance, per-vector results, and explicit indication of fallback usage. Strict mode (no-fallback) provides an implementation-only signal.

The SWP spec kit recommends publishing a conformance artefact bundle consisting of: (1) runner command lines, (2) runner stdout logs, (3) JSON summaries, and (4) the runner output schema documentation.

```
# Default run (fallback permitted)
make vectors SPEC_VECTOR_ARGS="-pattern 'conformance/vectors/core_*.json' -json-out artif
acts/conformance/core.default.json"

# Strict run (implementation-only; fallback disallowed)
make vectors-strict SPEC_VECTOR_ARGS="-pattern 'conformance/vectors/core_*.json' -json-ou
t artifacts/conformance/core.strict.json"

# Optional convenience targets (if provided by the repository)
make conformance-core
make conformance-all
```

Figure 2: Illustrative Artefact Bundle Commands

11. Deployment and Interoperability Scenarios

SWP is designed for incremental adoption. Representative scenarios include:

* _MCP bridging (C1)_: deploy SWP Core + MCP mapping behind a gateway that translates between MCP transports and SWP framing. MCP payload bytes remain opaque and can be forwarded byte-preservingly.

* _Federated delegation (C2)_: two organisations exchange agent tasks using SWP Core + A2A profile over an S1-compliant channel, with stable peer identity surfacing for authorisation and audit.

* _Unified RPC/events/artefacts (C3/C4/C5)_: a deployment uses SWP profiles for RPC, events, observability, and artifact transfer to carry task execution, progress, trace correlation, and artefact references on a single secured connection.

12. Limitations and Non-Goals

SWP Core is intentionally narrow. The following are explicit non-goals for this document:

* Replacing MCP or A2A; SWP is intended to carry and bridge them as profiles.

* Defining a new IAM system or full policy language; credential and policy profiles are propagation/signalling surfaces.

* Mandating a single transport; SWP runs over any channel meeting S1.

* Specifying message-level end-to-end cryptography across relays; S1 is a channel security binding.

* Providing Core-level multiplexing; multiplexing is transport- or profile-level.

This document is a Core substrate specification. Profile specifications define additional semantics and may impose further requirements for particular deployment environments.

13. Conclusion

SWP Core provides a minimal, independently implementable substrate for federated agent/tool communication: a rigorous record layer, a compact envelope with mandatory E1 encoding, explicit profile dispatch, and a conformance model based on classes, golden vectors, and reproducible artefact bundles. By separating a stable substrate from evolving application semantics (profiles), SWP aims to reduce bespoke gateway logic and make interoperability claims verifiable.

14. IANA Considerations

This document makes no requests of IANA.

15. References

15.1. Normative References

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

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

15.2. Informative References

[A2A] contributors, A. P. A., "Agent2Agent (A2A) Protocol", 2025, <https://a2a-protocol.org/>.

[A2A-BLOG] Blog, G. D., "A2A: A New Era of Agent Interoperability", 2025, <https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/>.

[FIPA-ACL] FIPA, "FIPA ACL Message Structure Specification", 2002, <https://www.fipa.org/specs/fipa00071/XC00071B.pdf>.

[GRPC] Authors, G., "gRPC", 2025, <https://grpc.io/>.

[GRPC-H2] Project, G., "gRPC over HTTP/2 Protocol", 2025, <https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md>.

[KQML] UMBC, "Knowledge Query and Manipulation Language (KQML)", 1997, <https://www.cs.umbc.edu/research/kqml/>.

[MCP] contributors, A. A., "Model Context Protocol (MCP) Specification", 2025, <https://modelcontextprotocol.io/specification/>.

[OTEL-PROP] Authors, O., "OpenTelemetry Context Propagation", 2025, <https://opentelemetry.io/docs/concepts/context-propagation/>.

[PROTOBUF] Google, "Protocol Buffers: Proto3 Language Guide", 2025, <https://protobuf.dev/programming-guides/proto3/>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018, <https://datatracker.ietf.org/doc/html/rfc8446>.

[RFC9113] Thomson, M. and C. Benfield, "HTTP/2", RFC 9113, June 2022, <https://datatracker.ietf.org/doc/html/rfc9113>.

[W3C-TRACE] W3C, "Trace Context", 2021, <https://www.w3.org/TR/trace-context/>.

Appendix: Implementation and Conformance Notes (Informative)

Implementations that claim SWP conformance should publish: (1) supported conformance classes, (2) artefact bundle outputs from default and strict runs, and (3) any deployment-specific policy parameters (size limits, timestamp freshness policy, peer authentication mode). The SWP spec kit includes runner output schema documentation and conformance claim checklists.

Author’s Address

Jericko Tejido
Email: jtbibliomania@gmail.com