IPPM Working Group
Internet-Draft
Updates: draft-melegassi-ippm-mvps-bundle-00 (if approved)
Intended status: Standards Track
Expires: 29 November 2026

L. Melegassi
Catellix

28 May 2026

A Forward-Compatible Extension and Capability-Negotiation Mechanism for the MVPS Bundle Format
draft-melegassi-ippm-mvps-extensions-00

Abstract

This document defines a forward-compatible extension point for the Multi-Vantage Path Synchrony (MVPS) bundle format. The base format already provides a schema version, a path-fingerprint version prefix, and an IANA "MVPS Bundle Capability Flags" registry, but its serialization is closed to unknown fields. This document adds a single, typed, namespaced "extensions" object and a processing rule (ignore-but-preserve), and specifies how extensions interact with the canonical hash and the MVPS Proof Envelope.

The mechanism is designed so that an extension can never become a tamper vector and can never alter the core coherence detector: the detection statistic D^2 is, by construction, a function of the core bundle only. Every property in this document is backed by a machine-checkable numerical receipt.

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 29 November 2026.

(https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

"How is it versioned and extended?" is among the first questions a reviewer asks of any wire format. The MVPS bundle format answers the versioning half (a schema-version string and a fingerprint version prefix) and reserves an IANA capability-flags registry, but its JSON serialization sets "additionalProperties": false and is therefore CLOSED to unknown fields. Three things were missing:

- a defined PLACE for an extension payload to live;
- a forward-compatibility PROCESSING RULE for consumers that do not understand an extension; and
- a specification of how extensions interact with the canonical hash and the MVPS Proof Envelope.

This document supplies all three with a single controlled open point and proves that the result is deterministic, tamper-evident, and safe with respect to the core coherence detector. It Updates the bundle format when approved.

2. Terminology

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

Core bundle: the bundle fields the detector reads (schema_version, bundle_id, destination, coordination_window, snapshots).

Core projection: the bundle with the "extensions" member removed.

Extension: a namespaced member of the "extensions" object.

3. What the Base Format Already Provides

For the avoidance of doubt, the following already exist in the base bundle format and are NOT introduced here:

- schema-version, a string matching "mvps-bundle-v[0-9]+";
- the path-fingerprint version prefix "v1|", with the rule that a future fingerprint algorithm MUST use a different prefix;
- the IANA "MVPS Bundle Capability Flags" registry, with policy Specification Required [RFC8126].

This document builds on those rather than replacing them.

4. The "extensions" Object

A bundle MAY contain one top-level member named "extensions" whose value is a JSON object. Each member key MUST be either:

- a reverse-DNS namespace matching the regular expression ^[a-z0-9-]+(\.[a-z0-9-]+)+$ (e.g. "com.example.geo"); or

- the name of a capability registered in the IANA "MVPS Bundle Capability Flags" registry.

Each member value is an opaque JSON value defined by the owning namespace or registered specification. A producer MUST NOT place extension data anywhere other than under "extensions"; the rest of the bundle remains closed ("additionalProperties": false), so "extensions" is the SINGLE admission point for new data.

The registry tells a consumer WHICH capabilities exist; the "extensions" object tells it WHERE the corresponding payload is.

5. Processing Rules (Ignore-but-Preserve)

A consumer that does not understand an extension namespace:

- MUST ignore that extension for the purpose of semantics (it MUST NOT alter the consumer’s decisions); and

- MUST preserve that extension byte-exactly if it re-serializes or relays the bundle, so that the canonical hash and any Proof Envelope binding remain valid.

A consumer MUST NOT reject a bundle solely because it carries an unknown extension namespace, provided the bundle is otherwise valid and the "extensions" keys satisfy Section 4.

Together these rules are the standard "ignore unknown, but preserve" forward-compatibility discipline; their composition is a hash-stable round-trip (Section 10).

6. Interaction with the Canonical Hash and Proof Envelope

Extensions ARE part of the canonicalized bundle. The canonical form is computed with JCS [RFC8785] over the WHOLE bundle, including "extensions"; the path-fingerprint of the base format is unchanged (it is computed over hops, not over extensions).

Consequences:

- Editing or removing any extension changes the canonical bytes and therefore the SHA-256 digest, so an extension cannot be silently stripped or altered without detection.
- When a bundle is bound by the MVPS Proof Envelope [I-D.melegassi-ippm-mvps-proof-envelope], its extensions are inside the bound Merkle leaf and inherit the same tamper-evidence and (optional) post-quantum signature protection.

A specification MAY define an extension as "fingerprint-affecting" only by also updating the fingerprint version prefix per the base format; absent that, extensions MUST NOT change the fingerprint.

7. Core-Detector Invariance

The coherence statistic D^2 is computed over the CORE PROJECTION of the bundle only. Therefore:

D^2(bundle) = D^2(core projection of bundle)

for every possible "extensions" content. An extension -- even one whose payload embeds a field named like a core field -- CANNOT move the detector. This is a design invariant, not a convention: a

conformant implementation MUST compute D^2 from the core projection and MUST NOT read "extensions" into the detector input.

This property is what makes the open point safe: extensions are a transport and audit feature, never a covert channel into a detection decision.

8. Versioning: Minor vs Major

- MINOR change: adding or removing an extension while keeping schema-version unchanged. Minor changes are forward-compatible by Section 5 and 7; a v1 consumer continues to operate.
- MAJOR change: changing schema-version (still matching "mvps-bundle-v[0-9]+"). A major change signals an incompatibility a v1 consumer MAY reject. Major changes are the mechanism for evolutions that the extension point cannot express (e.g. a change to a core field or to the fingerprint algorithm).

Implementations MUST distinguish the two by the schema-version string and MUST NOT treat a minor extension addition as a major break.

9. Capability Negotiation

A deployment advertises support for a capability by its registered flag; a bundle carries the corresponding data under the matching "extensions" namespace. Two properties hold:

- NON-INTERFERENCE: distinct namespaces are distinct object keys, so two extensions commute under JCS and cannot perturb each other or the core.
- MONOTONICITY: enabling a capability can only ADD behavior under its own namespace; an advertised capability whose namespace key is absent from a given bundle is a no-op for that bundle.

Capability negotiation is therefore additive and order-free, which keeps interoperability analysis tractable as the registry grows.

10. Worked Properties (Numerical Receipt)

The properties above are checked by scripts/validate_extension_negotiation.py (exit 0, 7/7), with the receipt at evidence/extension_negotiation_receipt.json:

T-EXT-CANON	JCS over bundle+extensions is insertion-order independent;
T-EXT-BIND	editing or stripping an extension flips the SHA-256 digest;

T-EXT-CORE-INV	D^2 is unchanged even under a malicious nested "metrics" injection inside an extension;
T-EXT-IGNORE	ignore-for-semantics + preserve-for-hash compose to a stable round-trip;
T-EXT-GATE	distinct namespaces commute; an unactivated flag is inert;
T-EXT-VERSION	minor vs major are syntactically distinguishable;
T-EXT-CLOSED	the legacy closed schema rejects extensions; the new schema admits only namespaced keys under "extensions".

11. Security Considerations

The central security property is core-detector invariance (Section 7): an extension cannot influence the coherence decision, so a malicious or buggy extension cannot cause a false alarm or suppress a real one through the detector. Extensions are inside the canonical hash (Section 6), so they cannot be silently inserted, altered, or stripped when the bundle is bound by the Proof Envelope.

An extension MAY carry sensitive data; producers SHOULD apply the same privacy handling as for core fields, and a redaction-style extension SHOULD commit to redacted content by hash rather than omitting it, so that the canonical binding remains verifiable.

The "extensions" key policy (Section 4) constrains the open surface to validated namespaces; implementations MUST reject bundles whose "extensions" keys do not match Section 4, to prevent the open point from degrading into an arbitrary free-form field.

12. IANA Considerations

This document adds no new registry. It clarifies that the existing "MVPS Bundle Capability Flags" registry (Specification Required) governs the names usable as keys of the "extensions" object, in addition to reverse-DNS namespaces. Registrations SHOULD state the namespace key, the payload schema, and whether the extension is fingerprint-affecting (Section 6).

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, June 2017.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

13.2. Informative References

[I-D.melegassi-ippm-mvps-bundle] Melegassi, L., "Multi-Vantage Path Synchrony Bundle Envelope and Vector Algebra", Work in Progress.

[I-D.melegassi-ippm-mvps-proof-envelope] Melegassi, L., "MVPS Proof Envelope with Post-Quantum Protection", Work in Progress.

Author’s Address

Leonardo Melegassi
Catellix
Brazil
Email: melegassi@catellix.com