Network Working Group
Internet-Draft
Intended status: Experimental
Expires: November 5, 2026

Y. Wang
Independent

May 4, 2026

JEP Conformance and Test Suite
draft-wang-jep-conformance-00

Status of This Memo

Copyright Notice

Table of Contents

JEP Conformance and Test Suite
Schemas, Validation Results, Test Vectors, and Reference Validator Behavior
draft-wang-jep-conformance-00

Author: Yuqiang Wang

Intended status: Experimental
Companion draft: draft-wang-jep-judgment-event-protocol-06

Abstract

This document defines conformance classes, validation-result structure, schema requirements, test-vector categories, reference-validator behavior, and implementation testing guidance for the Judgment Event Protocol (JEP). It is a companion to draft-wang-jep-judgment-event-protocol-06.

The purpose of this document is to make JEP implementations testable, interoperable, and auditable across languages, platforms, identity systems, and deployment profiles.

---

1. Introduction

JEP-Core defines event semantics and validation rules. Profiles define optional interoperability bindings. This document defines how implementations demonstrate conformance.

Conformance does not mean that an implementation has made any legal, policy, factual, or external-target determination. Conformance means that the implementation processes JEP events according to the declared JEP-Core version, conformance class, validation mode, and profile set.

---

2. Conformance Classes

2.1 JEP-Core-0.6 Producer

A producer conforms to JEP-Core-0.6 Producer if it can generate:

- I-JSON-compatible JEP events;
- required top-level fields;
- valid J/D/T/V verb-specific structures;
- UUIDv4 or profile-accepted nonce values;
- algorithm-tagged digest strings;
- detached signatures over JCS-canonicalized unsigned payloads;
- ext and ext_crit according to extension rules.

2.2 JEP-Core-0.6 Verifier

A verifier conforms to JEP-Core-0.6 Verifier if it can perform:

- duplicate member rejection;
- field-shape validation;
- JCS canonicalization;
- detached JWS verification under the required conformance algorithm set;
- event hash calculation;
- validation-level reporting;
- failure-code reporting;
- critical extension processing;
- acceptance validation;
- archival validation.

2.3 JEP-Core-0.6 Archive Verifier

An archive verifier supports archival validation and MUST NOT reject events solely because when is outside the current freshness window.

2.4 JEP-Chain-0.6 Verifier

A chain verifier supports reference resolution, event-hash checks, termination effects, cycle detection, and observed-log assumption reporting.

2.5 JEP-Extension-0.6 Processor

An extension processor can process registered extensions, reject unknown critical extensions, and detect critical extension conflicts.

2.6 JEP-Baseline-Ed25519-JWS-JCS-0.6

For compatibility with the -05 interoperability baseline, this conformance class requires support for:

- I-JSON-compatible JSON;
- JCS canonicalization;
- detached JWS Compact Serialization for single signatures;
- JOSE alg value Ed25519;
- algorithm-tagged sha256 digest strings;
- UUIDv4 nonce syntax;
- event hash calculation over the full signed event object.

This conformance class does not make Ed25519 the only algorithm allowed by JEP-Core semantics. It defines one interoperable baseline.

2.7 Profile-Specific Conformance

Profile-specific conformance is defined by the relevant profile draft, for example DID/VC, X.509, OAuth/OIDC, RATS, HJS Archive, or JAC Chain.

---

3. Validation Result Schema

A validation result object SHOULD include:

```
{
  "valid": true,
  "level": 3,
  "mode": "archival",
  "profile": "jep-core-0.6",
  "conformance_class": "JEP-Core-0.6 Verifier",
  "scopes": ["syntax", "cryptographic", "actor_binding", "chain_integrity"],
  "event_hash": "sha256:...",
  "warnings": [],
  "errors": []
}
```

3.1 Required Fields

- valid: boolean
- level: integer from 0 to 4
- mode: string
- profile: string
- warnings: array
- errors: array

3.2 Error Object

```
{
  "code": "ERR_REF_HASH_MISMATCH",
  "message": "The referenced event hash does not match the canonical payload.",
  "level": 3,
  "recoverable": false,
  "evidence": ["event_003", "event_007"]
```

```
}
```

3.3 Warning Object

```
{
  "code": "WARN_ARCHIVAL_ONLY_ALGORITHM",
  "message": "The signature algorithm is acceptable for archival validation but prohibite
d for new events.",
  "level": 1
}
```

---

4. JSON Schema Requirements

The conformance suite SHOULD provide schemas for:

- jep-event.schema.json
- jep-ref.schema.json
- jep-extension.schema.json
- jep-signature.schema.json
- jep-profile.schema.json
- jep-validation-result.schema.json

Schemas MUST NOT attempt to prove cryptographic validity. Schemas only validate structural and data-shape requirements.

4.1 Minimal jep-event.schema.json Seed

The following minimal schema is a seed schema for structural validation. It is not a substitute for cryptographic verification, trust-profile evaluation, extension processing, or policy validation.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://jep.org/schemas/jep-event-0.6.schema.json",
  "title": "JEP Event",
  "type": "object",
  "additionalProperties": false,
  "required": ["jep", "verb", "who", "when", "nonce", "sig"],
  "properties": {
    "jep": { "type": "string", "const": "1" },
    "verb": { "type": "string", "enum": ["J", "D", "T", "V"] },
    "who": { "type": "string", "minLength": 1 },
    "when": { "type": "integer" },
    "what": {
      "oneOf": [
        { "type": "string", "pattern": "^[a-z0-9-]+:[0-9a-f]+$" },
        { "type": "object" },
        { "type": "null" }
      ]
    },
    "nonce": { "type": "string", "format": "uuid" },
    "aud": { "type": "string" },
    "ref": {
      "oneOf": [
        { "type": "string", "pattern": "^[a-z0-9-]+:[0-9a-f]+$" },
        { "type": "object" },
        { "type": "null" }
      ]
    },
    "ext": { "type": "object" },
    "ext_crit": {
      "type": "array",
      "items": { "type": "string" },
      "uniqueItems": true
```

```
    },
    "sig": {
      "oneOf": [
        { "type": "string" },
        { "type": "object" }
      ]
    }
  }
}
```

---

5. Reference Validation Flow

A reference validator SHOULD implement the following pseudo-code:

```
validate_event(event, mode, trust_profile, expected_audience=None):
    parse JSON
    reject duplicate member names
    validate core field presence and types
    validate verb-specific requirements
    unsigned = event without sig
    canonical = JCS(unsigned)
    verify detached signature over canonical
    event_hash = hash(JCS(event including sig))
    resolve actor/key using trust_profile
    verify key bound to who
    if mode == acceptance:
        verify nonce freshness and replay cache
        verify timestamp window
        verify audience if expected
    validate ref syntax and resolve if available
    process all ext_crit extensions
    apply chain rules if requested
    apply policy rules if requested
    return validation_result
```

A validator MUST NOT return Level 4 unless an applicable policy profile has actually been evaluated.

---

6. Test Vector Categories

6.1 Valid Core Vectors

The conformance suite SHOULD include:

- valid/J-basic.json
- valid/D-basic.json
- valid/T-basic.json
- valid/V-basic.json
- valid/delegation-chain-valid.jsonl
- valid/verification-scope-valid.json
- valid/archive-validation-valid.json

6.2 Invalid Syntax Vectors

- invalid/invalid-json.json
- invalid/duplicate-member.json
- invalid/missing-required-field.json
- invalid/unknown-verb.json
- invalid/invalid-timestamp.json

6.3 Invalid Cryptographic Vectors

- invalid/invalid-signature.json
- invalid/signature-over-wrong-payload.json
- invalid/ref-hash-mismatch.json
- invalid/unsupported-signature-alg.json
- invalid/prohibited-signature-alg.json
- invalid/alg-key-type-mismatch.json
- invalid/canonicalization-version-unsupported.json

6.4 Invalid Replay and Freshness Vectors

- invalid/nonce-replay.json
- invalid/timestamp-out-of-window.json
- invalid/event-expired.json

6.5 Invalid Chain Vectors

- invalid/chain-broken.json
- invalid/cycle-detected.json
- invalid/terminated-delegation-reused.json
- invalid/delegation-scope-exceeded.json
- invalid/complete-log-assumption-unsatisfied.json

6.6 Invalid Extension Vectors

- invalid/unknown-critical-extension.json
- invalid/extension-schema-invalid.json
- invalid/extension-conflict.json
- invalid/verification-scope-overclaim.json

6.7 Canonicalization Vectors

- canonicalization/jcs-equivalent-payloads.json
- canonicalization/jcs-invalid-normalization.json
- canonicalization/full-signed-event-hash.json
- canonicalization/unsigned-signing-payload.json

6.8 Profile Vectors

- profiles/did-vc-valid.json
- profiles/did-vc-expired-credential.json
- profiles/x509-valid.json
- profiles/oauth-context-valid.json
- profiles/rats-attestation-valid.json
- profiles/hjs-archive-reference-valid.json
- profiles/jac-chain-valid.json

6.9 Signed Baseline Vectors

The repository form of this conformance suite SHOULD include signed Ed25519/JWS/JCS baseline vectors:

- valid/J-basic-signed.json
- valid/D-basic-signed.json
- valid/T-basic-signed.json
- valid/V-basic-signed.json
- valid/delegation-verification-termination-chain.jsonl
- valid/public-keys.json

Each signed vector SHOULD include:

- full signed event;
- canonical unsigned payload;
- event hash;
- expected validation result.

---

7. Seed Event Examples

7.1 Unsigned Judgment Event


```
{
  "jep": "1",
  "verb": "J",
  "who": "did:example:agent-789",
  "when": 1742345678,
  "what": "sha256:aa55ad4393538f14e6b4961de1a29216eed93517cb6c2631a56a5ee75edb3b7a",
  "nonce": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "aud": "https://platform.example.com",
  "ref": null
}
```


7.2 Validation Result for Signature Failure


```
{
  "valid": false,
  "level": 0,
  "mode": "acceptance",
  "profile": "jep-core-0.6",
  "scopes": ["syntax"],
  "event_hash": null,
  "warnings": [],
  "errors": [
    {
      "code": "ERR_SIGNATURE_INVALID",
      "message": "The detached JWS signature does not verify over the JCS-canonicalized u
nsigned payload.",
      "level": 1,
      "recoverable": false
    }
  ]
}
```


7.3 Validation Result for Policy Failure


```
{
  "valid": false,
  "level": 3,
  "mode": "policy",
  "profile": "jep-core-0.6+local-policy",
  "scopes": ["syntax", "cryptographic", "actor_binding", "chain_integrity"],
  "event_hash": "sha256:...",
  "warnings": [],
  "errors": [
    {
      "code": "ERR_POLICY_REJECTED",
      "message": "The event is cryptographically and chain-valid but rejected by local po
licy.",
      "level": 4,
      "recoverable": true
    }
  ]
}
```


7.4 Seed J/D/T/V Event Shapes

The following seed vectors define minimal event shapes. Signatures are placeholder values and are not intended to be cryptographically valid unless replaced by generated detached JWS values in a conformance suite.

7.4.1 J-basic

```
{
  "jep": "1",
  "verb": "J",
  "who": "did:example:agent-789",
  "when": 1742345678,
  "what": "sha256:aa55ad4393538f14e6b4961de1a29216eed93517cb6c2631a56a5ee75edb3b7a",
  "nonce": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "aud": "https://platform.example.com",
  "ref": null,
  "sig": "PLACEHOLDER_DETACHED_JWS"
}
```


7.4.2 D-basic


```
{
  "jep": "1",
  "verb": "D",
  "who": "did:example:human-123",
  "when": 1742345681,
  "what": {
    "claim": "delegate",
    "delegatee": "did:example:agent-789",
    "scope": "summarize-document",
    "constraints": ["no-external-send"],
    "context": "sha256:1111111111111111111111111111111111111111111111111111111111111111"
  },
  "nonce": "2b8c7d6e-1111-4abc-9def-123456789abc",
  "aud": "https://platform.example.com",
  "ref": "sha256:2222222222222222222222222222222222222222222222222222222222222222",
  "sig": "PLACEHOLDER_DETACHED_JWS"
}
```


7.4.3 T-basic


```
{
  "jep": "1",
  "verb": "T",
  "who": "did:example:human-123",
  "when": 1742345690,
  "what": {
    "claim": "terminate",
    "target": "sha256:3333333333333333333333333333333333333333333333333333333333333333",
    "termination_scope": "delegation"
  },
  "nonce": "3c9d8e7f-2222-4abc-9def-123456789abc",
  "aud": "https://platform.example.com",
  "ref": "sha256:3333333333333333333333333333333333333333333333333333333333333333",
  "sig": "PLACEHOLDER_DETACHED_JWS"
}
```


7.4.4 V-basic


```
{
  "jep": "1",
  "verb": "V",
  "who": "did:example:verifier-123",
  "when": 1742345700,
  "what": {
    "claim": "verified",
    "verification_scope": ["syntax", "cryptographic", "actor_binding"],
    "result": "pass"
  },
  "nonce": "4d0e9f80-3333-4abc-9def-123456789abc",
  "aud": "https://platform.example.com",
  "ref": "sha256:4444444444444444444444444444444444444444444444444444444444444444",
```

```
  "sig": "PLACEHOLDER_DETACHED_JWS"
}
```

---

8. Conformance Test Assertions

Each test vector SHOULD include:

```
{
  "name": "ref-hash-mismatch",
  "input": "invalid/ref-hash-mismatch.json",
  "mode": "archival",
  "profile": "jep-core-0.6",
  "expected_valid": false,
  "expected_level": 2,
  "expected_error": "ERR_REF_HASH_MISMATCH"
}
```

Test assertions SHOULD specify:

- input path;
- validation mode;
- trust profile;
- expected validity;
- expected highest completed level;
- expected warning codes;
- expected error codes;
- expected event hash, if applicable.

---

9. Reference CLI Behavior

A reference validator CLI SHOULD provide:

```
jep-validate event.json
jep-validate event.json --mode archival
jep-validate-chain chain.jsonl
jep-explain-failure event.json
jep-check-profile event.json --profile did-vc
jep-run-tests test-vectors/
```

CLI output SHOULD be a validation result object.

---

10. Implementation Requirements

A conforming implementation SHOULD document:

- supported JEP-Core version;
- supported conformance classes;
- supported signature algorithms;
- supported canonicalization versions;
- supported hash algorithms;
- supported profiles;
- supported extensions;
- failure-code coverage;
- test-vector pass report.

---

11. Interoperability Report Format

An implementation MAY publish an interoperability report:

```
{
  "implementation": "example-jep-validator",
  "version": "0.1.0",
  "jep_core": "0.6",
  "conformance_classes": [
    "JEP-Core-0.6 Verifier",
    "JEP-Core-0.6 Archive Verifier"
  ],
  "profiles": ["jep-profile:did-vc:0"],
  "tests_passed": 142,
  "tests_failed": 0,
  "date": "2026-05-04"
}
```

---

12. Security Considerations

A conformance suite can detect implementation divergence but cannot prove deployment security. Implementations remain responsible for secure key storage, algorithm policy, trust-profile correctness, replay-cache operation, and secure evidence storage.

Test keys MUST NOT be used in production.

---

13. Privacy Considerations

Test vectors SHOULD avoid real personal data. Profile test vectors that exercise identity, credential, or evidence features SHOULD use synthetic identifiers and non-sensitive sample claims.

---

14. Versioning

The conformance suite SHOULD version test vectors independently from JEP-Core. JEP-Core-0.6 can remain stable while conformance tests evolve as 0.6.1, 0.6.2, and so on.

Changes that alter JEP-Core validation semantics require a JEP-Core revision, not only a conformance-suite update.

---

15. Initial Repository Layout

Recommended repository layout:

```
jep-conformance/
  schemas/
    jep-event.schema.json
    jep-ref.schema.json
    jep-extension.schema.json
    jep-signature.schema.json
    jep-profile.schema.json
    jep-validation-result.schema.json
  test-vectors/
    valid/
    invalid/
    canonicalization/
    profiles/
  reference-validator/
    README.md
```

```
    pseudo-code.md
    validation-flow.md
  examples/
    basic-judgment-event.json
    delegation-termination-chain.jsonl
    verification-scope-example.json
    vc-backed-jep-event.json
  reports/
    interoperability-report.schema.json
```

---

16. Author’s Address

Yuqiang Wang
Email: signal@humanjudgment.org
URI: https://github.com/hjs-spec

17. Engineering-Complete Seed Package

The repository version of this draft set includes:

- seed JSON Schemas under schemas/;
- signed Ed25519/JWS/JCS baseline vectors under test-vectors/valid/;
- invalid examples under test-vectors/invalid/;
- canonicalization fixtures under test-vectors/canonicalization/;
- a minimal Python reference validator under reference-validator/.

These materials are intended to make JEP-Core-0.6 implementation-seeded. They are not a substitute for independent review, security audit, or production conformance testing.

18. Local CI and Test Harness

The repository version of this draft set includes a local test harness:

- Makefile
- pyproject.toml
- tests/test_reference_validator.py
- .github/workflows/conformance.yml
- test-manifest.json
- reports/local-validation-report.json

A conforming project MAY use these files as a bootstrap for continuous integration. Passing the seed suite does not imply production security or full profile conformance; it demonstrates baseline processing of signed JEP-Core-0.6 vectors and selected invalid cases.

19. Expanded Validation Artifacts

The repository version of this draft set includes:

- rendered Internet-Draft style .txt and seed RFCXML .xml files under ietf-rendered/;
- profile-specific test vectors under test-vectors/profiles/;
- additional invalid vectors for duplicate JSON members, nonce replay, terminated delegation reuse, extension conflict, algorithm downgrade, and expired credentials;
- TypeScript validator seed under typescript-validator/;
- verb-specific structural constraints in schemas/jep-event.schema.json.

v0.6 Direct Upgrade Note

This document is part of the JEP v0.6 draft set. The v0.6 set directly incorporates standardization structure, interoperability profiles, conformance artifacts, schemas, signed test vectors, invalid cases, and multi-language validator seeds without changing the JEP-Core narrow-waist semantics.