Internet Engineering Task Force
Internet-Draft
Intended status: Best Current Practice
Expires: 13 June 2026

A. Jurkovikj

10 December 2025

HTTP Profile for Synchronized Resource State (Agentic State Transfer)
draft-jurkovikj-httpapi-agentic-state-00

Abstract

HTTP resources are frequently exposed in multiple representations (e.g., text/html for rendering and application/json for processing) via Content Negotiation. However, standard HTTP concurrency controls (such as ETags) are typically scoped to the byte sequence of a specific representation. This creates a synchronization gap where a client modifying one representation cannot guarantee consistency with the underlying state of another representation, leading to race conditions and "lost updates" in multi-client environments.

This document specifies _Agentic State Transfer_ (AST), an HTTP profile for managing _Canonical Resource State_ (CRS) across multiple synchronized representations of the same resource. It defines the use of _Semantic Validators_ (Content Identity) to track the logical state of a resource independent of its serialization. It further mandates the use of Optimistic Concurrency Control via If-Match headers for state-changing operations, ensuring that mutations applied by automated agents are atomic and consistent across all views.

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 13 June 2026.

Table of Contents

1. Introduction

The architecture of the World Wide Web allows a single logical resource (identified by a URI) to be represented in multiple formats. A common pattern in modern systems is to expose:

* a _Rendered Representation_ (e.g., text/html) for human agents, and

* a _Structural Representation_ (e.g., application/json) for automated agents.

In AST terminology, rendered representations are typically _Projected Representations_, while a structural representation MAY serve as the _State-Bearing Representation (SBR)_ when it exposes the full Canonical Resource State.

HTTP provides mechanisms for content negotiation [RFC9110], and defines validators such as ETag for caching and conditional requests. However, in most deployments:

* validators are derived from a particular representation's bytes, and

* concurrency control, if present at all, is scoped to a single representation.

This creates a _synchronization gap_: two clients interacting with different representations of the same logical resource can unwittingly overwrite each other's work.

AST addresses this problem by treating the _Canonical Resource State_ (CRS) as the primary object of synchronization, and by standardizing how HTTP validators are used as _Semantic Validators_ for that CRS.

1.1. The Synchronization Problem

Consider a resource /article/123 exposed both as HTML and JSON.

1. _Client A (Browser)_ retrieves the HTML representation.
2. _Client B (Automated Agent)_ retrieves the JSON representation.
3. Client A submits an edit via an HTML form, updating the underlying database record.
4. Client B, unaware of the change, computes an update based on its stale JSON representation and submits a PUT or PATCH request.

In typical HTTP APIs, at least one of the following is true:

* there is no ETag at all;
* the ETag is tied to the bytes of a specific representation (e.g., JSON only); or
* the server does not enforce preconditions (If-Match).

As a result, Client B's request may succeed, even though:

* its view of the resource is stale, and
* the update logically conflicts with Client A's change.

This is the classic _Lost Update_ problem, now exacerbated by the growing use of automated agents acting as first-class editors.

1.2. Canonical Resource State (CRS)

To address this, AST introduces the concept of _Canonical Resource State_ (CRS):

> CRS is the authoritative, logical state of the resource, independent of any particular representation or serialization.

For example, the CRS might be:

* a database row (or set of rows),
* a document or AST (abstract syntax tree),
* a domain object in application memory, or

* in a WordPress CMS, the post record in wp_posts with fields like post_content, post_title, post_status, and associated metadata.

Representations such as HTML, JSON, or Markdown are derived _from_ the CRS and are considered projections of that state.

AST defines a profile where:

1. *Shared Identity* -- All HTTP representations of an AST resource are projections of the same CRS.

2. *Semantic Validation* -- Validators (ETag values) are derived from the CRS, not a specific representation’s bytes, so that a change to the CRS invalidates all representations together.

3. *Mandatory Concurrency* -- State-changing operations on AST resources MUST provide a precondition (If-Match) based on the CRS validator.

This enables safe interaction by automated agents and human clients, ensuring they participate in the same state transition model.

2. 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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals.

Canonical Resource State (CRS) The underlying, authoritative logical state of a resource (e.g., the database record or abstract syntax tree). The CRS is the atomic unit of concurrency.

State-Bearing Representation (SBR) A specific representation (e.g., application/json) that exposes the CRS in a lossless, deterministic format. The SBR serves as the Concurrency Lock for the resource. Its ETag represents the version of the CRS.

Projected Representation A representation derived from the CRS for a specific consumption pattern (e.g., text/html for rendering, text/csv for analysis). Projected Representations are Read-Only with respect to the CRS; they reflect the state but do not govern it.

Semantic Validator The Strong ETag associated with the SBR. It changes if and only if the CRS changes.

AST Resource A resource that conforms to this profile.

3. The AST Profile

This section defines the Agentic State Transfer (AST) profile. It specifies how clients synchronize state across multiple representations by pivoting on the State-Bearing Representation (SBR).

3.1. Representation Roles

For a given AST Resource, the server MUST designate exactly one representation as the State-Bearing Representation (SBR). All other representations are considered Projected Representations.

* The SBR acts as the source of truth for state synchronization.

* Projected Representations are derivative views. While they MAY have their own caching validators (ETags) based on their specific byte sequences, they MUST NOT be used as the basis for state-changing preconditions on the CRS.

3.2. Semantic Validators (The SBR ETag)

The server MUST assign a Strong ETag to the SBR. This ETag serves as the Semantic Validator for the resource.

* *Determinism*: The SBR ETag MUST change whenever the CRS changes.

* *Scope*: The SBR ETag MUST NOT be reused as the ETag for Projected Representations, as this would confuse standard HTTP caches.

Canonicalization: Implementations MAY compute the SBR ETag from a deterministic canonical serialization of the CRS (e.g., JSON Canonicalization Scheme, RFC 8785 [RFC8785]). This ensures that logically identical states produce identical validators, regardless of field ordering or formatting variations. See Section 4.1 for detailed guidance.

3.2.1. ETag Computation (Informative)

The SBR ETag MUST be a strong validator as defined in [RFC9110]. Implementations MAY compute the SBR ETag using any method that satisfies both the determinism and strong validator requirements. Common approaches include:

* *Content Hash*: SHA-256 or similar cryptographic hash over the canonicalized SBR bytes (e.g., "sha256-YWJjMTIz..."). This is RECOMMENDED for interoperability, as it allows clients to verify state equivalence without server coordination and naturally produces strong validators.

* *Version Counter*: Monotonic version number from the data store (e.g., "v42" or "rev-1234"). This is simple but requires centralized version tracking. The version must increment on every CRS change to maintain strong validator semantics.

The content hash approach provides the strongest guarantee that identical CRS values produce identical ETags across different server instances or time periods, which aids debugging and distributed deployments.

3.3. Discovery (rel="state")

To enable clients consuming a Projected Representation (e.g., HTML) to discover the SBR for synchronization or editing, servers SHOULD include a Link header with the state relation type. The type attribute MUST be present to indicate the media type of the SBR, enabling clients to determine how to interact with it.

Example (response for GET /article/123.html):

http HTTP/1.1 200 OK Content-Type: text/html ETag: "html-v1" Link: </api/article/123>; rel="state"; type="application/json"

3.4. Read Semantics

3.4.1. Reading the SBR

Clients SHOULD use the SBR ETag for conditional requests (If-None-Match) on the SBR URI. The server MUST return 304 Not Modified if the CRS has not changed. On a 304 response, the server SHOULD include the current SBR ETag in the response headers.

Clients MUST NOT use the SBR ETag as If-None-Match against Projected endpoints. Projected endpoints have their own byte-specific validators.

Example: HEAD Request to Fetch SBR Validator

Clients MAY use HEAD to obtain the current SBR ETag without transferring the full representation body:

‘‘‘http HEAD /api/article/123 HTTP/1.1 Host: example.com Accept: application/json

HTTP/1.1 200 OK Content-Type: application/json ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F" Content-Length: 4521 ‘‘‘

This allows clients to check for state changes with minimal bandwidth usage before deciding whether to fetch the full representation.

3.4.2. Reading Projections

Clients MAY cache Projected Representations using their specific ETags. However, clients MUST recognize that a Projected Representation may be stale relative to the CRS if the SBR ETag has changed.

3.5. Write Semantics (The Safe Write Protocol)

To modify the Canonical Resource State, clients MUST perform the mutation against the SBR Endpoint (or an endpoint explicitly bound to the SBR’s validator).

Clients *MUST NOT* use a Projected Representation’s ETag as an If-Match precondition for CRS mutations. Only the SBR’s own ETag is valid for concurrency control.

1. *Precondition*: The client obtains the current SBR ETag (via GET or HEAD on the SBR URI).

2. *Mutation*: The client sends the state-changing request (PUT, POST, PATCH) including the If-Match header set to the SBR ETag.

3. *Verification*:

    * If the If-Match header is missing, servers SHOULD respond with 428 Precondition Required [RFC6585]. Servers MAY respond with 400 Bad Request if 428 is not supported by the implementation environment, provided the error body clearly indicates the missing precondition requirement.

    * If the provided ETag does not match the current CRS version, the server MUST respond with 412 Precondition Failed.

    * If the provided ETag matches the current CRS version, the server MUST accept the write (subject to authorization), update the CRS, and invalidate all Projected Representations.

Note: For endpoints that do *not* implement AST semantics, servers MAY accept unconditional writes as in traditional HTTP APIs. See the "Non-AST Resources and Fallback Behavior" section for guidance on mixed deployments.

4. *Response*: Upon a successful write, the server SHOULD return the new SBR ETag in the response headers, allowing the client to chain subsequent updates immediately.

5. *Error Bodies*: For 412, 428, and 400 (when used for missing preconditions) responses, servers SHOULD return a Problem Details body [RFC7807] to assist automated clients in resolving the conflict.

Example: 428 Response for Missing Precondition

When a client attempts a state-changing operation without If-Match, the server SHOULD return a 428 response:

‘‘‘http PATCH /api/article/123 HTTP/1.1 Host: example.com Content-Type: application/json

HTTP/1.1 428 Precondition Required Content-Type: application/problem+json

{ "type": "https://example.com/errors/precondition-required", "title": "Precondition Required", "status": 428, "detail": "If-Match header is required for CRS mutations" } ‘‘‘

Alternative: 400 Response When 428 Is Not Available

If the server environment does not support 428, it MAY use 400 with a clear error message:

‘‘‘http PATCH /api/article/123 HTTP/1.1 Host: example.com Content-Type: application/json

HTTP/1.1 400 Bad Request Content-Type: application/problem+json

{ "type": "https://example.com/errors/missing-precondition", "title": "Missing Required Precondition", "status": 400, "detail": "If-Match header is required for state-changing operations on AST resources" } ‘‘‘

Example: 412 Response with Problem Details

When a client’s If-Match precondition fails, the server SHOULD return a Problem Details response indicating the current state:

```
‘‘‘http PATCH /api/article/123 HTTP/1.1 Host: example.com Content-
Type: application/json If-Match:
"sha256-ABC123OLDVALUE456789STALE012"

HTTP/1.1 412 Precondition Failed Content-Type: application/
problem+json ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F"

{ "type": "https://example.com/errors/precondition-failed", "title":
"Precondition Failed", "status": 412, "detail": "The resource state
has changed since you last read it.", "current-etag":
"sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F", "provided-etag":
"sha256-ABC123OLDVALUE456789STALE012" } ‘‘‘
```

The inclusion of the current ETag in the error response allows the client to immediately refetch the latest state without an additional round trip.

3.5.1. Conflict Resolution (Informative)

When a client receives a 412 Precondition Failed response, it SHOULD follow one of these resolution strategies:

1. *Refetch and Retry*: Fetch the current SBR to obtain the new ETag and state, then reapply the intended mutation and retry with the updated ETag.

2. *Three-Way Merge*: If the client maintains the original state it read, the changes it intended to make, and can obtain the current server state, it MAY attempt an automatic merge (similar to version control systems). This is only appropriate when changes affect non-overlapping fields or can be deterministically combined.

3. *User Intervention*: Present the conflict to the user (human or automated decision system) for manual resolution, showing both the client’s intended changes and the server’s current state.

4. *Abort*: Abandon the update if the conflict indicates the client’s view is too stale or the changes are incompatible.

Servers MAY assist conflict resolution by including the current state or a conflict diff in the 412 response body (see Problem Details example above). This reduces round trips and provides clients with immediate context for resolution.

Retry Example:

```
‘‘‘http # Client receives 412 HTTP/1.1 412 Precondition Failed
Content-Type: application/problem+json ETag:
"sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F"
```

4. Client immediately refetches

```
GET /api/article/123 HTTP/1.1 If-None-Match:
"sha256-ABC123OLDVALUE456789STALE012"

HTTP/1.1 200 OK ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F" {
...current state... }
```

5. Client reapplies mutation with fresh ETag

```
PATCH /api/article/123 HTTP/1.1 If-Match:
"sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F" ‘‘‘
```

5.1. SBR Endpoint Headers

To ensure correct caching and transfer behavior, SBR endpoints SHOULD include the following headers:

* *Cache-Control*: SBR responses SHOULD include Cache-Control: no-transform to prevent intermediaries from modifying the representation (which would invalidate the Semantic Validator).

* *Vary*: If the SBR supports compression or other content transformations, servers SHOULD set Vary: Accept-Encoding to ensure caches distinguish between compressed and uncompressed variants.

* *Accept-Ranges*: SBR endpoints SHOULD include Accept-Ranges: none if range requests would break the semantic integrity of the representation (e.g., partial JSON structures).

Example SBR Response Headers:

```
http HTTP/1.1 200 OK Content-Type: application/json ETag:
"sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F" Cache-Control: no-cache, no-
transform Vary: Accept-Encoding Accept-Ranges: none Content-Digest:
sha-256=:RK63H5GH1GQJQM8A7LN7RP8VNR7FVXWH1TQKJM8QZH4=:
```

5.2. Caching Considerations

5.2.1. SBR Caching

SBR endpoints SHOULD follow these caching practices:

* *Non-Negotiated SBR*: Prefer a single canonical media type for the SBR to avoid cache fragmentation.

* *Strong ETag*: The SBR MUST expose a strong ETag as the Semantic Validator.

* *Cache-Control*: Use Cache-Control: no-cache, no-transform to require revalidation while allowing storage. If the SBR contains sensitive or user-specific data, use Cache-Control: private or no-store as appropriate.

* *Vary*: Set Vary: Accept-Encoding if compression is applied to allow caches to distinguish between compressed and uncompressed variants.

* *Accept-Ranges*: Consider Accept-Ranges: none to prevent partial-byte range requests that could create ambiguity in state validation.

* *304 Revalidation*: Support conditional requests with If-None-Match and respond with 304 Not Modified when the SBR ETag matches.

* *ETag Scope*: Do NOT reuse the SBR ETag on Projected endpoints, as this would conflate semantic and byte-level validation.

5.2.2. Projected Representation Caching

Projected Representations SHOULD be cached using standard HTTP caching mechanisms:

* *Representation-Specific ETags*: Use their own ETags based on the byte content of each projection.

* *Standard Cache-Control*: Apply appropriate Cache-Control directives (e.g., max-age, public, private) based on the projection’s volatility and sensitivity.

* *Client-Side Invalidation*: Clients that perform CRS mutations SHOULD invalidate their own cached Projected Representations after a successful write or upon receiving a 412 Precondition Failed response. Note that shared caches and intermediaries will not automatically invalidate projections based on SBR changes.

6. Computing and Managing Semantic Validators

AST does not mandate a single algorithm for computing Semantic Validators. However, interoperability and predictability benefit from consistent approaches.

6.1. Canonicalization

A common pattern is to derive the Semantic Validator from a canonical serialization of the CRS. For example:

1. Serialize the CRS to a structured format (for example, JSON).
2. Canonicalize the serialization (for example, by sorting keys and controlling numeric formats).
3. Compute a hash over the canonical form (for example, SHA-256).
4. Encode and prefix the hash to produce the ETag value.

Servers *SHOULD* ensure that:

* fields that are purely operational or volatile (for example, last access timestamps) are excluded from the canonical form; and
* the canonicalization process is deterministic across deployments.

Servers MAY use [RFC8785] JSON Canonicalization Scheme (JCS) or an equivalent deterministic procedure.

6.2. Storage Models

Servers MAY:

* compute Semantic Validators on demand from the CRS; or
* store precomputed Semantic Validators alongside the CRS.

When validators are stored:

* they *SHOULD* be updated atomically with the CRS; and
* they *MUST* be invalidated or recomputed whenever the CRS changes.

When validators are computed on demand, implementations *SHOULD* ensure that the cost of computing the validator is small relative to the cost of generating the representation.

6.3. Relationship to Application Backends

AST is agnostic to how applications store and manage CRS. For example:

* a SQL application might treat a row (or set of rows) as CRS, with the Semantic Validator derived from a version column and relevant fields;

* a document store might treat a document body plus metadata as CRS;

* a CMS might treat a block tree or content graph as CRS.

The only requirement is that the Semantic Validator change whenever the logical CRS changes, regardless of how many representations exist.

6.4. Non-AST Resources and Fallback Behavior

Servers MAY implement AST for some resources and not others.

For resources that do not follow AST:

* servers MAY continue to use representation-specific ETags as byte validators;

* servers MAY accept unconditional writes (no If-Match required), as in traditional HTTP APIs; and

* clients MUST NOT assume AST semantics.

Servers SHOULD clearly document which resources implement AST.

7. Integrity and Transport

In standard HTTP usage, ETags often serve a dual purpose: validating the semantic state for caching/concurrency, and validating the integrity of the response bytes (to detect transport corruption).

In the AST profile, the State-Bearing Representation (SBR) ETag serves as the *Semantic Validator* for the Canonical Resource State (CRS). Projected Representations MAY have their own representation-specific ETags for caching purposes, but these MUST NOT be used as Semantic Validators for concurrency control.

7.1. Separation of State and Integrity

To allow clients to verify the integrity of the payload bytes independently of the semantic state, AST servers *SHOULD* use the Content-Digest header field defined in [RFC9530].

* *ETag (SBR)*: validates the _Canonical Resource State_ (logical truth) and is used with If-Match and If-None-Match for concurrency control.

* *ETag (Projected)*: MAY be used for caching individual representations, but MUST NOT be used for state-changing preconditions.

* *Content-Digest*: validates the _representation bytes_ (wire truth) and is used to verify that the transfer completed successfully.

Clients *MUST NOT* send If-Match or If-None-Match preconditions using a Projected Representation’s ETag when interacting with the SBR endpoint. Only the SBR’s own ETag participates in concurrency control logic.

Example Response (SBR):

http HTTP/1.1 200 OK Content-Type: application/json ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F" Content-Digest: sha-256=:RK63H5GH1GQJQM8A7LN7RP8VNR7FVXWH1TQKJM8QZH4=:

Example Response (Projected):

http HTTP/1.1 200 OK Content-Type: text/html ETag: "html-v1" Link: </api/article/123>; rel="state"; type="application/json" Content-Digest: sha-256=:X8Z9Y7W6V5U4T3S2R1Q0P9O8N7M6L5K4=:

This separation allows intermediaries and clients to detect transport corruption without conflating it with state changes.

In AST, the SBR ETag serves as a semantic validator for the Canonical Resource State. Clients SHOULD use Content-Digest for verifying byte integrity of the transfer. Outside of AST contexts, traditional ETag usage for caching and integrity remains unchanged.

8. Discovery of Synchronized Representations

To enable agents to discover different representations of the same AST resource, servers *SHOULD* expose relationships between representations using the HTTP Link header [RFC8288].

8.1. The state Relation (SBR Discovery)

As defined in Section 3.3, Projected Representations SHOULD include a Link header with rel="state" pointing to the State-Bearing Representation (SBR). This allows clients consuming a Projected Representation to discover the SBR for synchronization or editing.

The state relation identifies the authoritative SBR for concurrency control purposes. Clients MUST use the ETag from the state target (the SBR) for all If-Match and If-None-Match preconditions on state-changing operations.

Example (Projected Representation response):

http HTTP/1.1 200 OK Content-Type: text/html ETag: "html-v1" Link: </api/article/123>; rel="state"; type="application/json"

8.2. The alternate Relation (General Cross-Representation Navigation)

The alternate link relation type is *RECOMMENDED* for general navigation between synchronized representations of the same AST resource.

When providing an alternate link:

* the type attribute *MUST* be present to indicate the media type of the target representation (for example, application/json, text/html); and

* the profile attribute *MAY* be used to indicate a specific schema or capability set of the target representation.

Example (SBR response linking to Projected):

http HTTP/1.1 200 OK Content-Type: application/json ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F" Link: </article/123.html>; rel="alternate"; type="text/html"

Note: The state relation specifically identifies the SBR for concurrency control purposes. In contrast, rel="alternate" is for general cross-representation navigation and does *not* imply concurrency semantics. Clients MUST NOT assume that a representation linked via alternate participates in the AST concurrency protocol.

8.3. Bidirectional Linking

To ensure agents can navigate between representations regardless of their entry point, linking *SHOULD* be bidirectional.

* If Representation A links to Representation B, then Representation B *SHOULD* link back to Representation A (or to a canonical URI that resolves to A).

This ensures that an agent encountering a structural representation (for example, JSON) can identify the corresponding rendered representation (for example, HTML) for attribution, verification, or human inspection.

8.4. Profile Advertisement

To enable clients to detect whether a resource implements AST semantics, servers SHOULD advertise this using the profile link relation [RFC6906] via one of the following mechanisms:

Link Header with Profile Relation:

http HTTP/1.1 200 OK Content-Type: application/json Link: <https://datatracker.ietf.org/doc/draft-jurkovikj-http-agentic-state/>; rel="profile" ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F"

Content-Type Profile Parameter:

http HTTP/1.1 200 OK Content-Type: application/json; profile="https://datatracker.ietf.org/doc/draft-jurkovikj-http-agentic-state/" ETag: "sha256-RK63H5GH1GQJQM8A7LN7RP8VNR7F"

This allows clients to detect AST resources and apply appropriate concurrency control workflows (If-Match preconditions, 412 handling, etc.) instead of assuming traditional HTTP semantics.

9. Security Considerations

This profile relies on existing HTTP security mechanisms. However, the synchronization of multiple representations introduces specific security considerations.

9.1. Consistency and Race Conditions

The primary security benefit of AST is the prevention of race conditions (Lost Updates). By enforcing If-Match checks against the CRS, the server ensures that an agent cannot overwrite state changes made by another client, even if those changes were made via a different representation.

Applications *SHOULD* treat failed preconditions (412, 428) as an expected outcome in concurrent editing scenarios and provide appropriate conflict resolution mechanisms.

9.2. Information Leakage via Divergence

If the access control policies for different representations diverge, there is a risk of information leakage.

* Servers *MUST* ensure that the authorization requirements for accessing a structural representation (for example, JSON) are at least as strict as those for the rendered representation (for example, HTML).

* Servers *MUST* ensure that a structural representation does not expose sensitive data that is redacted in the rendered representation, unless the client is explicitly authorized to see the raw state.

If different audiences see different subsets of the CRS, the server SHOULD either:

* maintain separate CRS domains with separate Semantic Validators; or

* apply view-specific filtering consistently to all representations exposed to that audience.

9.3. Validator Guessing

Because Semantic Validators are often derived from content (for example, a hash), they may be susceptible to offline guessing attacks if the content has low entropy.

Implementations *SHOULD* ensure that:

* validators have sufficient entropy; or

* validators are combined with a secret salt; or

* validators are derived from internal versioning mechanisms rather than directly from plaintext content,

if the content version itself is sensitive.

Validators MUST NOT be treated as authentication or authorization tokens.

10. IANA Considerations

10.1. Link Relation Type Registration

This document requests registration of the "state" link relation type in the Link Relation Type Registry defined by [RFC8288]:

* *Relation Name*: state
* *Description*: Refers to the State-Bearing Representation (SBR) of the current resource for concurrency control. The target is the authoritative CRS view and exposes the strong ETag used for If-Match and If-None-Match on CRS-affecting operations.
* *Reference*: This document (Sections 3.3 and 6.1)
* *Notes*: The target's media type SHOULD be suitable for machine processing (e.g., application/json). Clients MUST use the target's strong ETag (not the source representation's ETag) as the validator for CRS preconditions. The type attribute SHOULD be present on the link to indicate the target's media type.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012.

[RFC9530] Polli, R. and L. Pardue, "Digest Fields", RFC 9530, DOI 10.17487/RFC9530, February 2024.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016.

[RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013.

11.2. Informative References

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020.

[RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013.

[RFC7386] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7386, DOI 10.17487/RFC7386, October 2014.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016.

Appendix A. Appendix A. Implementation Status

Note to RFC Editor: Please remove this section before publication.

This section records the status of known implementations of the AST profile at the time of publication. Based on [RFC7942], reviewers are invited to report implementation experiences.

A.1. WordPress Dual-Native Implementation

Organization: Independent implementation

Description: A WordPress plugin implementing the AST profile for content management and template synchronization.

Implementation Details:

* *SBR Endpoint*: Exposes WordPress posts and templates as application/json representations at /wp-json/dual-native/v1/posts/{id} and /wp-json/dual-native/v1/design/templates/...

* *Semantic Validators*: Uses SHA-256 content hashes as strong ETags, formatted as "sha256-{base64}", computed from canonicalized JSON representation of the CRS.

* *Concurrency Control*: Enforces If-Match preconditions on all state-changing operations (POST, PATCH). Returns 412 Precondition Failed on ETag mismatches and 428 Precondition Required when If-Match is missing.

* *Projected Representations*: HTML rendering at canonical WordPress URLs with rel="state" Link headers pointing to the SBR endpoint.

* *Discovery*: Implements rel="state" link relations on HTML responses and rel="alternate" on SBR responses.

* *Multi-Bot Testing*: Demonstrated in production with multiple concurrent agents. Testing showed 89.4% bandwidth savings for monitoring workloads and zero lost updates (vs. 67% lost with standard WordPress REST API).

Maturity: Production use since November 2025.

Coverage: Implements all normative requirements of this specification including SBR designation, strong ETag generation, If-Match enforcement, 412/428 error handling with Problem Details (RFC 7807), and Link header discovery.

Licensing: Open source

Contact: antunjurkovic@gmail.com

Author’s Address

Antun Jurkovikj
Email: antunjurkovic@gmail.com