Reliable Context

RELIABLE CONTEXT / RESEARCH & PRACTICAL GUIDANCE

JSON-LD context changes: When does data meaning change?

> Reliable Context is a kernel of truth: a system of record for organizations that fuse private and public information and must defend every value they act on. One canonical record per real thing, a cited source and an observation date behind every value, unknowns declared rather than left blank, and any past state of the record reproducible.

Published by Reliable Context · 2026-10-13 · json-ld context changes · json-ld · context versioning · semantic drift · json-ld validation · remote contexts · data provenance · knowledge engineering · semantic compatibility

Original article: https://reliablecontext.com/articles/json-ld-context-changes-data-meaning



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes
4. Defining a payload-level compatibility contract
5. Freezing remote contexts and verifying integrity
6. Implementation Considerations and Process Changes
7. Releasing contexts with provenance and migration evidence
8. Data Analysis and Evidence
9. Implications and Future Directions
10. Frequently Asked Questions (FAQs)
11. Conclusion

Executive Summary

JSON-LD (JavaScript Object Notation for Linked Data) context changes alter data meaning when the same payload acquires different identifiers, relationships, literal interpretations, or graph placement after processing. A context connects local JSON terms to Internationalized Resource Identifiers (IRIs); expansion makes those mappings explicit. (niem.github.io) The World Wide Web Consortium (W3C) JSON-LD 1.1 Recommendation dates to **July 16, 2020**. (www.w3.org) Its existence does not provide a universal compatibility classification for subsequent changes to an organization's contexts. The practical decision therefore requires a declared exchange contract, representative payloads, fixed context artifacts, and named consumers.

This report proposes a **payload-level semantic regression test**. Freeze the old and proposed dependency bundles, expand each fixture under controlled options, compare the expanded representations, then evaluate the consumers' required behavior. Resource Description Framework (RDF) comparison must account for blank-node renaming: RDF/JS documents graph-structure equality with normalized blank nodes, while Apache Jena provides an isomorphism operation that ignores their labels. (rdf.js.org) (jena.apache.org) Keep expanded-document comparison where consumers rely on index or direction information that ordinary RDF conversion can lose. (www.w3.org)

The proposed release classifications are **unchanged**, **intended breaking**, **accidental breaking**, and **unknown**. Protected terms restrict redefinition, with exceptions for identical definitions and property-scoped contexts. (www.w3.org) They are one input to the proposed test. Likewise, a weak Hypertext Transfer Protocol (HTTP) entity tag can remain unchanged when representation data changes; a retained expected digest provides a different check on content identity. (www.rfc-editor.org) (csrc.nist.gov) **National Information Exchange Model (NIEM)** guidance recommends treating contexts as vetted exchange artifacts that can be cached instead of depending on runtime network resolution. (niem.github.io)

The quantitative example below is **illustrative**: across **five synthetic fixtures**, **20 baseline assertions** include **four replacements**, or **20%**, despite an unchanged assertion count. These are constructed values, not processor benchmarks or deployment measurements. In a concrete context-versioning example, SEMICeu's

maintainer issue **#103**, opened **September 25, 2025** (github.com), was marked completed **May 8, 2026**. (github.com) Its closure is evidence about that discussion, not a general compatibility guarantee. Release a context as compatible only within the tested contract; version and migrate deliberate changes; hold unexplained differences or unavailable evidence.

Introduction and Background

A JavaScript Object Notation (JSON) document can remain byte-for-byte identical while its interpretation changes. Consider a key called **owner**. One context can map it to one relationship IRI, while another maps it to a different relationship. Alternatively, its key mapping can stay constant while its string value changes from a literal to a reference identifying another node. The official expansion suite includes a fixture specifically verifying that identifier coercion produces a node reference. (w3c.github.io) A text diff of the payload cannot reveal a change that resides in the interpretation artifact.

The necessary terms are precise. A **context** is the mapping environment that connects case-sensitive terms to IRIs and supplies other interpretation settings. (www.w3.org) An **IRI** is the identifier used here for a graph node, property, or datatype; compact names are abbreviations whose expansion depends on the context. NIEM's guide explains local-name aliasing to IRIs. (niem.github.io) **Expansion** produces the explicit identifier and value representation used in this report's comparisons. NIEM illustrates differently named documents giving the same expanded result. (niem.github.io)

Term coercion instructs a processor how to interpret values associated with a term, such as references rather than strings or explicitly typed literals. The official fixtures distinguish identifier coercion from native-value datatype coercion. (w3c.github.io) A **protected term** restricts redefinition. (www.w3.org) A **remote context** is a context obtained through a referenced document; custom loaders can control document and context retrieval. (pypi.org) A **consumer**, for this report, is a named application or processing path whose behavior the release promises to preserve.

The proposed method addresses context engineers exchanging graph-shaped records. It treats the artifact's interpretation as part of the evidence needed to defend an exchanged value. Reliable Context's public system-of-record description includes a cited, verbatim, dated source behind every value; that observable guarantee motivates retaining the interpretation artifacts used for an exchange. This is a release-evaluation method proposed by this report.

Structural validation remains useful. JSON Schema describes JSON structure, constraints, and data types. (json-schema.org) However, an organization's schema acceptance criteria and its expanded-meaning acceptance criteria should be tested separately. A valid JSON shape can accompany an intentionally or accidentally changed context.

Key Changes

Identifier mappings and fallback vocabulary

The first change class concerns **which thing or relationship a term identifies**. A property mapping can change, a type name can acquire another IRI, or a local alias can be introduced. NIEM demonstrates the converse: differently named JSON documents can evaluate to the same expanded data under appropriate contexts. (niem.github.io) Consequently, changed spelling does not establish changed meaning, and unchanged spelling does not establish preserved meaning.

The **@vocab** setting supplies fallback mappings for properties and types without matching explicit definitions. (www.w3.org) Testing only terms visibly edited in a context can therefore miss a wider effect. The recommended fixture inventory includes both explicitly mapped keys and keys resolved through fallback vocabulary. Record the resulting predicate and type IRIs, rather than inferring them from the apparent similarity of namespace strings.

Treat identifier migration as a contract change when consumers rely on the former identifiers. Two different IRIs should remain different in the comparison unless the contract explicitly authorizes another equivalence relation. A proposed migration can separately establish that a consumer recognizes both identifiers. That additional behavior requires evidence; it should not be silently assumed by the comparator.

Coercion, language, datatype, and collection interpretation

A context can alter **what a value represents**. Identifier coercion expands a value into a node reference, while datatype coercion adds a datatype to an expanded value. (w3c.github.io) A default language applies to strings in the official language fixture. (w3c.github.io) Schema.org's documentation provides a concrete implementation perspective: its context is generated automatically to distinguish default URL and text interpretations for property strings. (schema.org)

The following is a **Hypothetical Example** with synthetic identifiers. It shows one unchanged payload evaluated under two separately supplied contexts. For each evaluation, supply the corresponding `@context` value shown below through the `expandContext` option when expanding the payload:

```
{
  "@id": "https://example.invalid/record/alpha",
  "owner": "https://example.invalid/person/ada"
}
```

Old context:

```
{
  "@context": {
    "owner": {
      "@id": "https://example.invalid/vocab/owner",
      "@type": "@id"
    }
```

```
  }
}
```

Proposed context:

```
{
  "@context": {
    "owner": "https://example.invalid/vocab/owner"
  }
}
```

Expected old expansion:

```
[
  {
    "@id": "https://example.invalid/record/alpha",
    "https://example.invalid/vocab/owner": [
      {"@id": "https://example.invalid/person/ada"}
    ]
  }
]
```

Expected proposed expansion:

```
[
  {
    "@id": "https://example.invalid/record/alpha",
    "https://example.invalid/vocab/owner": [
      {"@value": "https://example.invalid/person/ada"}
    ]
  }
]
```

The old object identifies a node; the proposed object holds a string literal. These expected expansions illustrate the documented coercion behavior, rather than report execution of a processor. (www.w3.org) The recommendation is to classify this change as breaking for a consumer requiring an owner relationship to another node.

Collection comparison needs its own rule. Ordinary JSON-LD arrays do not express element order, whereas lists do. (www.w3.org) The official expansion manifest consequently distinguishes ordinary array comparison from ordered list comparison. (w3c.github.io) Sorting every array before comparison can erase a list change that matters to consumers.

Scope, imports, and protection

A context's effects depend on where it applies. **Scoped contexts** apply through properties or types; type scopes ordinarily stop at the next node. (www.w3.org) The same local key should therefore be exercised under different applicable scopes rather than checked only at the root.

@import introduces another artifact into context processing, with wrapping entries taking precedence when merged. (www.w3.org) Retaining the wrapper alone is insufficient for a reproducible evaluation of its imported definitions.

Protected terms are a useful defense against prohibited redefinitions, with important exceptions. (www.w3.org) A passing protected-term test should be reported alongside expansion and consumer results. Organizations can add profile-specific restrictions to the declared contract. The proposed harness should exercise those restrictions explicitly and retain the resulting outcomes alongside the general expansion comparison.

Defining a payload-level compatibility contract

Compatibility is relative to an explicit set of inputs and consumers. Semantic Versioning itself requires a declared public application programming interface (API) before version labels can communicate compatibility. (semver.org) For a JSON-LD exchange, the proposed public contract includes accepted payloads, interpretation artifacts, processor settings, output invariants, and consumer expectations.

The evaluation should declare:

- **Payload population:** Which record shapes, scopes, and value categories the fixtures represent.
- **Consumers:** Which importer, query view, validator, or transformation path receives the records.
- **Identifier invariants:** Which subjects, predicates, object references, and types must remain stable.
- **Value invariants:** Which datatypes, languages, directions, and literal representations are significant.
- **Graph invariants:** Whether graph membership and blank-node structure are preserved.
- **Operational outcomes:** Which processing failures, dropped properties, and consumer results are accepted.
- **Coverage boundary:** Which payloads, consumer versions, and remote dependencies remain untested.

These are proposed evaluation fields. RDF/JS documents datasets as unordered sets, supporting a distinction between dataset content and serialization order. (rdf.js.org) Its data-model specification also states that validation of supplied values is outside that specification, so constructing RDF terms alone should not be treated as a complete validation procedure. (rdf.js.org)

Table 1 below defines the proposed release decision rule. The table is an analytical framework, not a classification prescribed by JSON-LD.

Class	Observed evidence within the contract	Release decision
Unchanged	Expanded meaning and required consumer results match; dependency evidence is complete.	Release as compatible for the stated tested boundary.

Class	Observed evidence within the contract	Release decision
Intended breaking	Documented differences match an approved migration; affected consumers have a transition path.	Publish a new context version and migration evidence.
Accidental breaking	Differences or processing errors contradict the intended change.	Hold release and repair the candidate or revise the intended contract.
Unknown	A required artifact, fixture, comparison, or consumer result is unavailable.	Hold the compatibility claim until evidence is supplied.

The distinction between breaking and unknown prevents a missing test from being presented as a successful result. Semantic Versioning associates incompatible API changes with a major increment, but applying that convention to contexts requires identifying the relevant contract first. (semver.org) A version label communicates a decision; it does not perform the comparison.

Use **two comparison layers** where appropriate. Expanded-document comparison preserves the JSON-LD interpretation under the declared options. Dataset isomorphism evaluates graph structure after a defined RDF conversion. RDF/JS documents normalization of blank nodes, and Jena distinguishes structural isomorphism from equality of underlying Java graph objects. (rdf.js.org) (jena.apache.org) These APIs should be selected deliberately, since object identity and structural equivalence answer different questions.

Freezing remote contexts and verifying integrity

A reproducible comparison needs a **closed dependency bundle** for each candidate. NIEM's guidance treats contexts as information-exchange specification artifacts that should be inspected, vetted, and cached. (niem.github.io) The recommended acceptance harness extends that principle to every context resource actually used during processing.

For each side of the comparison, retain:

- **Root artifact:** The exact old or proposed context bytes and release identifier.
- **Dependency inventory:** Every referenced or imported context used by fixtures.
- **Retrieval identity:** Requested Uniform Resource Locator (URL), resolved document URL, and observation time.
- **Expected digest:** The algorithm, retained digest, and exact byte representation covered.
- **Loader policy:** The URLs permitted and the response returned for each.
- **Processor options:** Processing mode, base handling, and any initial expansion context.
- **Fixture identity:** Payload bytes, fixture identifier, and source or construction note.
- **Run evidence:** Processor identity, outcomes, expansion outputs, and comparison findings.

These retention fields are proposed controls. JSON-LD's API provides a custom document loader and an initial expansion-context option. (www.w3.org) PyLD documents an allowlist-based FrozenDocumentLoader that rejects other URLs, providing one implementation mechanism for restricting retrieval. (pypi.org) A local-file loader is another mechanism, but the Rust FsLoader documentation explicitly says loaded documents are not cached; the current file can therefore remain a changing input. (docs.rs)

Cache validation and byte integrity should have separate evidence fields. Request for Comments (RFC) 9111 describes entity tags as cache validators and explains that a successful conditional validation can authorize reuse of a stored response. (www.rfc-editor.org) RFC 9110 distinguishes strong validators, which change with observable representation data, from weak validators that need not change for every data change. (www.rfc-editor.org) Retain the context itself rather than treating a cache-validation event as an archival substitute.

The National Institute of Standards and Technology (NIST) describes secure hash digests as a means of detecting changes since digest generation. (csrc.nist.gov) The recommended harness computes against an independently retained expected digest and rejects mismatches. Also preserve the bytes: a digest can identify a difference without providing the old artifact needed to reproduce an earlier expansion.

Content addressing offers another identity mechanism. InterPlanetary File System (IPFS) documentation defines a content identifier (CID) as addressing content itself and explains that a CID includes codec information as well as a hash. (docs.ipfs.tech) Identical files can have different CIDs when encoding and construction choices differ. (docs.ipfs.tech) A CID-based bundle should therefore specify those choices. This report does not require IPFS; its documentation illustrates why the artifact-identity scheme must be defined.

Finally, preserve resolution metadata. The API's base setting and the special handling of **@base** in remote contexts mean that relative-reference behavior cannot be inferred solely from the context's visible text. (www.w3.org) The proposed test reproduces the original resolution environment and records intentional differences in the candidate environment.

Implementation Considerations and Process Changes

Separate generation, validation, expansion, and comparison

Generation is an upstream step. **LinkML** documents generating JSON-LD contexts from models, with an option to use class or slot compact identifiers as keys. (linkml.io) Its generation-date stamping is disabled by default for reproducible output. (linkml.io) A generator change therefore belongs in the evidence bundle whenever it changes the candidate artifact.

Structural validation should then run independently. JSON Schema validation evaluates an instance against a schema, while **Ajv** generates validation code according to that schema and caches compiled schema functions. (json-schema.org) (ajv.js.org) A passing structural result is an input to the release review. The separate expansion comparison establishes whether the same accepted structure acquires different graph-shaped meaning.

Compaction is useful for inspection but should not be the sole comparator. **RDFLib** demonstrates both auto-compacted serialization and serialization with a supplied custom context. (rdflib.readthedocs.io) Those examples show why a compact representation can change with presentation choices. A regression report should retain expanded outputs so reviewers can inspect the identifiers and values behind the compact names.

The recommended execution sequence is:

- **Validate fixtures:** Check structural constraints and document excluded inputs.
- **Resolve bundles:** Ensure every required context is present and digest-checked.
- **Expand the baseline:** Use the retained old bundle and recorded options.
- **Expand the candidate:** Use the proposed bundle with deliberate option differences identified.
- **Compare representations:** Preserve list ordering and relevant literal distinctions.
- **Compare datasets:** Apply declared RDF conversion and blank-node isomorphism rules.
- **Exercise consumers:** Evaluate named outputs and record the final release classification.

The official expansion manifest supplies comparison conventions, including attention to list ordering and language-tag normalization. (w3c.github.io) These are useful starting points, but consumer-specific equality criteria should be stated separately.

Include negative fixtures and missing-information detection

A positive fixture verifies a successful interpretation. Negative fixtures verify expected errors and explain whether the harness can distinguish a prohibited context change from a retrieval problem. The API specifies errors for protected-context nullification and protected-term redefinition. (www.w3.org) It also distinguishes invalid import values, invalid remote contexts, retrieval failures, and nested imports. (www.w3.org)

The recommended negative set includes:

- **Protected collision:** A later context attempts a prohibited term redefinition.
- **Context clearing:** A null context encounters protected definitions.
- **Unavailable dependency:** A required remote context cannot be supplied.
- **Malformed context:** Retrieved content lacks the required context structure.
- **Import conflict:** An imported context contains a prohibited nested import.
- **Mode conflict:** A declared processing mode conflicts with the context version.

The official negative fixture for explicit JSON-LD 1.0 processing against a 1.1 context establishes that the last case is a real conformance category. (w3c.github.io) These fixtures should record expected errors rather than inflate a success percentage by treating every completed call as successful.

Also examine **dropped properties**. Expansion can ignore properties that do not map to usable identifiers. PyLD documents a dropped-property handler that can raise an exception when this happens. (pypi.org) A completed expansion is therefore insufficient evidence that every input value survived. Compare the set of meaningful input fields with the interpreted output, using expected omissions documented in each fixture.

The official JSON-LD site distinguishes its browser-based Playground from test suites used to validate processors. (json-ld.org) Use interactive inspection to understand an example and automated fixtures to preserve the acceptance decision.

Releasing contexts with provenance and migration evidence

A release packet should connect the context artifacts, the evaluation run, and the resulting decision. The **provenance (PROV) Data Model** distinguishes entities with fixed aspects, activities acting on entities, and agents bearing responsibility. (www.w3.org) Applying those concepts here is a modeling choice: retained

context versions are entities, expansion executions are activities, and responsible software or maintainers can be agents.

Dublin Core provides complementary descriptive terms for modification dates, derivation sources, and custody-related provenance. (www.dublincore.org) The proposed packet records these relationships alongside the artifacts themselves. Metadata that points to missing bytes does not satisfy the report's reproducibility criterion.

For an intended breaking release, publish:

- **Change rationale:** Which interpretation is deliberately changing and why.
- **Artifact identities:** Old and proposed root contexts and dependency bundles.
- **Affected fixtures:** Payload classes and the exact changed identifiers or value interpretations.
- **Consumer scope:** Applications, versions, and acceptance tests involved.
- **Migration instructions:** Rewrites, dual-reading periods, or consumer changes required.
- **Unknowns:** Untested scopes, blocked dependencies, and unavailable results.
- **Reproduction recipe:** Inputs, options, expected outputs, and decision evidence.

Semantic Versioning prohibits changing the contents of an already released version. (semver.org) That convention supports publishing another versioned artifact rather than overwriting the interpretation needed by existing records. PROV also distinguishes a version-specific identifier from a latest-version resource whose state can change. (www.w3.org) Treat the convenient latest URL and the retained artifact identity as separate fields.

The distinction matters in real documentation. Schema.org describes dated vocabulary release snapshots, while its workflow page separately states that context-document history is not archived. (schema.org) These are statements about the documented artifacts; a vocabulary snapshot should not automatically be substituted for a captured context dependency.

The SEMICeu maintainer issue is a narrow example of explicit context-versioning discussion. Its author proposed versioning to track changes, especially breaking changes, when opening issue **#103 on September 25, 2025**. (github.com) The issue records closure as completed **on May 8, 2026**. (github.com) The linked implementation commit was not examined for this report, so the issue's closure does not establish a particular implemented compatibility policy. It supplies maintainer evidence, not market-demand measurement.

Table 2 below turns the proposed fixture requirements into a review checklist.

Fixture category	Include in the payload set	Evidence to retain
Identifier mapping	Explicit terms, aliases, and vocabulary fallback cases.	Expanded subject, predicate, object, and type identifiers.
Value interpretation	References, ordinary strings, typed values, and language-tagged strings.	Expanded reference or value object and datatype or language.
Scope and dependencies	Nested nodes, property scopes, type scopes, and imports.	Applicable context path and complete dependency bundle.

Fixture category	Include in the payload set	Evidence to retain
Collections and graphs	Ordinary arrays, ordered lists, blank nodes, and named graphs.	Comparison rules, graph membership, and structure results.
Negative behavior	Protected collisions, unavailable contexts, malformed imports, and mode conflicts.	Expected error, actual error, and consumer handling.
Consumer behavior	Records needed by each declared downstream application.	Named consumer result and accepted migration outcome.

This checklist is intended to expose coverage gaps. The official suite already distinguishes identifier, datatype, language, and processing-mode behavior, while graph-comparison documentation supplies separate structural equality concepts. (w3c.github.io) (rdf.js.org) A fixture library assembled around those distinctions can support a bounded compatibility claim without suggesting exhaustive coverage.

Data Analysis and Evidence

Illustrative statement-change counts

The following **Hypothetical Example** uses constructed assertions rather than collected deployment data. Each fixture has an identified subject and **four baseline assertions**. The candidate replaces one assertion in each of the first four fixtures and preserves the final fixture. All subjects are identified, so this small calculation has no blank-node matching ambiguity.

Define the comparison unit as a subject, predicate, object, and graph tuple. Literal objects additionally carry their declared datatype and language. For this example, a replacement pairs a removed baseline assertion with an added candidate assertion by fixture subject and field. This pairing rule is declared before counting; it would need refinement for multivalued properties or differently shaped graphs.

Table 3 below contains the illustrative counts. Every number in it is an author-constructed example, not an independently measured processor result.

Synthetic fixture	Baseline assertions	Preserved	Replaced	Illustrative change
Predicate mapping	4	3	1	A relationship uses a different predicate IRI.
Reference coercion	4	3	1	A referenced node becomes a string literal.
Default language	4	3	1	A string receives a different language annotation.
Graph membership	4	3	1	An assertion moves to another graph.
Unused alias	4	4	0	The added term is unused in this fixture.
Total	**20**	**16**	**4**	Candidate also contains 20 assertions.

The **baseline replacement rate is 4 / 20 = 20%**. Separately, a set-difference report would contain **four removals and four additions**, yielding **eight difference entries**. Dividing those eight entries by the 20 baseline assertions produces **40%**, which is a different metric. The report should name the numerator,

denominator, and matching method rather than present either percentage as an undefined semantic-change rate.

Cardinality alone would show no difference because both sides contain **20 assertions**. The example demonstrates the analytical limitation of a count-only comparator. It does not estimate how frequently real context changes alter facts.

Evidence boundaries and coverage

Published documentation establishes the relevant mechanisms, not a measured acceptance rate for an organization's contexts. The official expansion suite covers node-reference coercion, native-value datatype annotation, default language application, and processing-mode conflict. (w3c.github.io) PyLD documents regression testing and synchronized specification suites, but that description is not an independent measurement of a particular installation. (pypi.org)

Report **fixture coverage** separately from **change prevalence**. Record which mapped terms, scope paths, value forms, dependency paths, and consumer outputs have exercised fixtures. A term inventory alone is insufficient: the same term can participate in different value and scope situations. The proposed packet identifies missing coverage directly and classifies unresolved required cases as unknown.

The recommended results record includes baseline and candidate outcomes, comparison-unit counts, removed and added assertions, approved replacements, dropped input fields, and consumer acceptance. RDF/JS's unordered dataset model and Jena's blank-node-aware isomorphism operation illustrate why ordering and blank-node label changes should be evaluated with explicit comparison rules. (rdf.js.org) (jena.apache.org)

No production throughput, latency, prevalence, or commercial-demand estimate is inferred here. Such estimates require their own datasets, sampling methods, and measurements. A reproducible small fixture result answers a bounded release question; it should not be extrapolated into an organization-wide accuracy claim.

Implications and Future Directions

The central engineering implication is that **the interpretation artifact belongs in the record's evidence boundary**. A retained payload without its applicable context dependencies may be insufficient for reproducing the exchanged interpretation. NIEM's artifact-based guidance and the documented frozen-loader mechanism provide practical foundations for the proposed bundle. (niem.github.io) (pypi.org)

Integrity checks and provenance records should remain distinguishable. RFC 9530 defines Content-Digest over actual message content and a separate representation digest mechanism. (www.rfc-editor.org) It explicitly excludes authentication from its defined scope, and its security discussion explains that content and its accompanying digest can be replaced together. (www.rfc-editor.org) The recommendation is to retain the approved expected digest through the organization's trusted release process and record who approved the artifact. A matching digest proves agreement with that retained expectation, not the correctness of the context's interpretation.

Provenance also needs an executable evidence boundary. A derivation description can express an update from one entity to another, but replay requires the actual inputs and declared processing environment. (www.w3.org) Repositories for storing and managing provenance documents, such as the one described by OpenProvenance, serve a complementary documentary role. (openprovenance.org) The acceptance packet should link that documentation to the retained contexts, fixtures, and outputs.

For a system of record, this connects interpretation history to record history. Reliable Context's public description says that past record states can be reproduced. An organization's JSON-LD exchange policy can evaluate the analogous observable property: can a reviewer reproduce the interpretation used for this record at this observation time? That question can be tested without assumptions about a product's internal implementation.

Future evaluations should distinguish context changes from processor upgrades and vocabulary changes. Run those factors separately where possible, then explicitly test the combinations intended for release. A context-compatible result under one processing environment is a scoped conclusion. The proposed practice is to publish those boundaries alongside the decision, so future consumers can determine whether the original evidence applies to their own processing paths.

Frequently Asked Questions (FAQs)

How does a JSON-LD context affect data meaning?

It supplies term-to-identifier mappings and interpretation settings. The same local name can therefore identify a different property, and identifier coercion can turn a string into a node reference. (niem.github.io) (w3c.github.io) Compare expanded outputs for unchanged payloads to expose those differences.

Does JSON-LD context validation establish compatibility?

Structural validation and processor conformance address useful but narrower questions. JSON Schema specifies JSON constraints, and the official JSON-LD test suites validate processors. (json-schema.org) (json-ld.org) The proposed compatibility test additionally evaluates old-versus-candidate interpretation and the declared consumers.

How should remote context changes be detected?

Retain the approved dependency bundle, check the fetched or supplied bytes against retained expected digests, and compare fixture expansion results under controlled loading. NIEM recommends cached exchange artifacts, and PyLD documents a loader restricted to an explicit URL allowlist. (niem.github.io) (pypi.org) Detecting a byte change and determining its semantic effect are separate results.

Do protected terms prevent semantic drift?

They restrict certain redefinitions, with specified exceptions. (www.w3.org) They do not replace the wider fixture comparison proposed here. Include scopes, imports, fallback mappings, coercion, and consumer behavior in the declared coverage.

How should JSON-LD contexts be versioned?

Declare the exchange contract, classify the observed change, and preserve released artifacts. Semantic Versioning provides public-API and immutable-release conventions that can be applied to that contract. (semver.org) A changed context needs a new artifact identity even when its tested expanded meaning is unchanged.

Can expanded documents be compared as ordinary JSON?

Use JSON comparison only with documented JSON-LD-aware rules. The official expansion manifest distinguishes general array handling from ordered lists, and graph-level comparison additionally needs blank-node handling. (w3c.github.io) (rdf.js.org) Preserve relevant expanded metadata when RDF conversion would discard it. (www.w3.org)

Conclusion

A context change is an interpretation change until the declared evidence establishes otherwise. The recommended release decision starts with representative payloads and named consumers, retains the old and proposed context dependency bundles, and compares expansion under recorded processing conditions. It then tests graph structure and consumer behavior according to explicit equality rules.

The proposed classifications keep the decision legible. Unchanged means the required invariants and consumer results match within the tested boundary. Intended breaking means the observed differences match a documented migration. Accidental breaking means the candidate contradicts its stated intent. Unknown means required evidence is unavailable. None of those labels should conceal the tested population or the untested cases.

The illustrative counts show why equal statement totals cannot establish equal meaning. The synthetic candidate retains the same total while replacing relationships, reference interpretation, language annotation, or graph membership. A useful evidence packet names the comparison unit, reports additions and removals, and declares the denominator for any percentage.

For context engineers, the durable deliverable is the **reproducible acceptance packet**: artifacts, dependency identities, fixtures, options, expanded outputs, comparison results, and consumer outcomes. Version labels and protected terms contribute to that packet, but the release decision rests on the observable interpretation of exchanged records. Publish compatible changes with their scope, publish deliberate changes with migration evidence, and hold cases whose meaning or coverage remains unresolved.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://niem.github.io/json/reference/json-ld/context
2. https://www.w3.org/TR/json-ld11
3. https://rdf.js.org/dataset-spec
4. https://jena.apache.org/documentation/javadoc/jena/org.apache.jena.core/org/apache/jena/rdf/model/Model.html

5. https://www.rfc-editor.org/rfc/rfc9110.html
6. https://csrc.nist.gov/pubs/fips/180-4/upd1/final
7. https://github.com/SEMICeu/LinkedDataEventStreams/issues/103
8. https://w3c.github.io/json-ld-api/tests/expand-manifest
9. https://pypi.org/project/PyLD
10. https://json-schema.org/learn/getting-started-step-by-step
11. https://schema.org/docs/howwework.html
12. https://www.w3.org/TR/json-ld11-api
13. https://semver.org
14. https://rdf.js.org/data-model-spec
15. https://docs.rs/json-ld/latest/json_ld/loader/struct.FsLoader.html
16. https://www.rfc-editor.org/rfc/rfc9111.html
17. https://docs.ipfs.tech/concepts/content-addressing
18. https://linkml.io/linkml/generators/jsonld-context.html
19. https://ajv.js.org/guide/getting-started
20. https://rdflib.readthedocs.io/en/stable/apidocs/examples.jsonld_serialization
21. https://json-ld.org
22. https://www.w3.org/TR/prov-dm
23. https://www.dublincore.org/specifications/dublin-core/dcmi-terms
24. https://www.rfc-editor.org/rfc/rfc9530.html
25. https://openprovenance.org

THE PUBLISHER

About Reliable Context

Reliable Context is a system of record for organizations that fuse private and public information and must defend every value they act on.

The guarantees

One canonical record per real thing: duplicates resolve by rule, ambiguity is shown rather than hidden, and related organizations relate without ever being merged. A verbatim, cited source and an observation date sit behind every value. Unknowns are declared as not found, blocked or not applicable with a reason, never left as an empty cell. Any past state of the record is reproducible, together with what changed since.

Where it runs

It runs inside the customer's perimeter, one database per organization. Permitted sources, model routing, reader clearance and retention are set as the customer's own policies, and every value is exportable with its chain of custody for an auditor or regulator.

What it is not

It is not a vector database or a RAG framework, not a data catalog, MDM or data-quality suite, not a CRM or data-enrichment vendor, and not a chatbot or AI assistant. Where a reader needs one of those, the honest answer is to say so and point them there.

Visit

See Reliable Context.

Reliable Context is in production use in two sectors; more than six thousand of its identity decisions were made by rule and none has been reversed. Articles published here explain general practice, cite their primary and academic sources, and name the jurisdiction of any regulatory rule they describe. They are not legal, regulatory or compliance advice for a specific organization, and requirements change.

Website: https://reliablecontext.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.