LMD: Linked Markdown — Markdown as Semantic Infrastructure

Ethan Davidson (ethan@wazoo.dev)

2026

Contents

1 Abstract **2**

2 Introduction **3**

3 LMD Design **3**
3.1 Zero Custom Syntax . . . 3
3.2 Document Model . . . 3
3.3 Linking . . . 4
3.4 Validation and Inference . . . 4
3.5 Query and Publishing . . . 4

4 Prior Art and Related Work **4**

5 Implementations **5**
5.1 Python Implementation . . . 5
5.2 TypeScript Implementation . . . 5

6 Discussion **5**
6.1 Limitations . . . 5
6.2 Adoption Path . . . 6

7 Future Work **6**

8 Conclusion **6**

9 References **6**

10 Appendix A: LMD Specification **6**

11 Linked Markdown (LMD) **6**
11.1 Status of This Document . . . 7
11.2 1. Introduction . . . 7
11.2.1 1.1. Design Goals . . . 7
11.2.2 1.3. Prior Art and Related Work . . . 7
11.2.3 1.4. Protocol Status and Versioning . . . 8
11.3 2. Conformance . . . 8
11.3.1 2.1. Document Conformance . . . 8
11.3.2 2.2. Processor Conformance . . . 8
11.3.3 2.3. Shape Conformance . . . 8
11.4 3. The LMD Document Model . . . 9
11.4.1 3.1. Document Identity . . . 9

11.4.2 3.2. Document Type . . . 9
11.4.3 3.3. The JSON-LD Context . . . 9
11.4.4 3.4. Vocabulary Conventions . . . 9
11.4.5 3.5. Body Content . . . 9
11.4.6 3.6. Links as Properties . . . 10
11.5 4. Frontmatter as JSON-LD . . . 10
11.5.1 4.1. Syntax . . . 10
11.5.2 4.2. Required Fields . . . 10
11.5.3 4.3. Recommended Fields . . . 10
11.5.4 4.4. CURIE Resolution . . . 11
11.5.5 4.5. Relation to RDF . . . 11
11.6 5. Document Linking . . . 11
11.6.1 5.1. Intra-Corpus Links . . . 11
11.6.2 5.2. External Links . . . 11
11.6.3 5.3. Fragment Identifiers . . . 11
11.7 6. Validation . . . 11
11.7.1 6.1. SHACL Validation . . . 11
11.7.2 6.2. JSON Schema Integration . . . 11
11.7.3 6.3. Shape Location . . . 12
11.8 7. Inference . . . 12
11.8.1 7.1. OWL-RL Deductive Reasoning . . . 12
11.8.2 7.2. Custom Axioms . . . 12
11.8.3 7.3. Opt-Out . . . 12
11.9 8. Query . . . 12
11.9.1 8.1. SPARQL 1.1 . . . 12
11.9.2 8.2. Embedded Query Blocks . . . 12
11.9.3 8.3. Result Formats . . . 13
11.109. Serialization . . . 13
11.10.19.1. RDF Export . . . 13
11.10.29.2. JSON-LD Export Specifics . . . 13
11.1110. Publishing . . . 13
11.11.110.1. Static HTML . . . 13
11.11.210.2. URL Structure . . . 13
11.11.310.3. Content Negotiation . . . 13
11.1211. Provenance . . . 13
11.12.111.1. Process Stamps . . . 13
11.12.211.2. PROV-O Alignment . . . 14
11.1312. Security Considerations . . . 14
11.13.112.1. IRI Injection . . . 14
11.13.212.2. Schema Loading . . . 14
11.13.312.3. SPARQL Injection . . . 14
11.1413. IANA Considerations . . . 14
11.15Appendix A: Complete LMD Document Example . . . 14
11.16Appendix B: LMD Namespace . . . 15
11.17Appendix C: Glossary . . . 15
11.18Appendix D: References . . . 16

1 Abstract

LMD (Linked Markdown) is a specification for structuring, validating, and querying typed Markdown documents as first-class semantic graph nodes. An LMD document is simultaneously valid CommonMark and valid JSON-LD — rendering in any standard Markdown renderer while participating in an RDF graph with SHACL validation, OWL-RL inference, and SPARQL query capability. No custom syntax is introduced; the

protocol lives entirely in frontmatter and linking conventions. This paper presents LMD's design, compares it with related approaches (Cagle 2026; ozekik 2023; davay42 2026; iunera 2025), and describes the TypeScript and Python reference implementations. The full specification is included as an appendix.

2 Introduction

Markdown has become the de facto standard for writing across software, documentation, knowledge management, and publishing. GitHub, Obsidian, Notion, VS Code, Pandoc, and the CommonMark specification have made `.md` files universal. However, Markdown documents remain opaque to semantic tooling: they are strings, not typed entities with known properties and relationships.

Existing approaches to adding semantics to Markdown fall into two camps: inline annotation languages (e.g., Markdown-LD using Turtle in body text (ozekik 2023), MD-LD using `{=iri}` syntax (davay42 2026)) and inference-based extraction (e.g., json-ld-markdown, which infers Schema.org JSON-LD from document structure (iunera 2025)). Both approaches have limitations: inline annotations introduce nonstandard syntax that can cause unpredictable rendering, while inference loses author-intent semantics.

LMD takes a different approach: **frontmatter is JSON-LD**. The protocol requires no custom syntax, no new file extension, and no special renderer. A single `@id` and `@type` field in the frontmatter turns any Markdown document into a typed RDF node. From this foundation, LMD layers validation (SHACL), inference (OWL-RL), query (SPARQL), and publishing — each capability independently adoptable by conforming processors.

3 LMD Design

3.1 Zero Custom Syntax

LMD introduces no nonstandard Markdown syntax. All protocol semantics are expressed through:

- Standard JSON-LD 1.1 frontmatter delimited by `---`
- Standard CommonMark links for document-level relationships
- Convention-based file organization (shapes, axioms, corpus structure)

A minimal LMD document is:

```
---
"@id": https://example.org/docs/my-item
@type: schema:Article
@context:
  schema: https://schema.org/
---

# My Item
Content here.
```

This file is valid Markdown, valid YAML, and valid JSON-LD simultaneously.

3.2 Document Model

Every LMD document is identified by a canonical IRI (`"@id"`) and one or more RDF types (`@type`). The `@id` and `@type` keywords are also valid in YAML frontmatter. Frontmatter fields map directly to RDF predicate-value pairs with the document's `@id` as subject. The Markdown body (everything after the frontmatter) is addressable as an RDF literal, typically via `schema:articleBody`.

Documents form a **corpus**: a collection of LMD documents sharing a configuration, a shapes directory (for SHACL validation), and optional axioms (for OWL-RL inference).

3.3 Linking

Intra-corpus links use standard Markdown link syntax `[text](target.md)`. A processor resolves the target filename to the target document's `@id` IRI. External links (to IRIs outside the corpus) are preserved as typed RDF object properties. Fragment identifiers (`#section-2`) may be typed as `rdf:HTML` content.

3.4 Validation and Inference

LMD delegates validation to **SHACL 1.1** (Knublauch and Kontokostas 2017). Shapes are loaded from a `shapes/` directory in the corpus. Each shape targets one or more document types via `sh:targetClass` and defines property constraints (`sh:path`, `sh:datatype`, `sh:minCount`, etc.).

Shapes may also reference JSON Schema via `lmd:jsonSchema` for deep structural validation of nested frontmatter.

Inference follows **OWL-RL** (Motik et al. 2012) deductive reasoning — enabling subclass reasoning, property chain expansion, and domain/range inference. Axioms are loaded from an `axioms/` directory. Processors may allow clients to opt out of inference.

3.5 Query and Publishing

Corpora are queryable via **SPARQL 1.1** (Harris and Seaborne 2013). Documents may embed SPARQL queries in fenced code blocks (`sparql`). Processors render query results inline when generating output.

LMD processors may publish corpora as static HTML sites, with content negotiation supporting HTML (browsers), JSON-LD (agents), and raw Markdown (per Cloudflare Markdown for Agents convention).

4 Prior Art and Related Work

System	Approach	Syntax	RDF Compat
DataBooks (Cagle 2026)	Markdown as semantic infrastructure design pattern	YAML frontmatter	Partial (not native JSON-LD)
Markdown-LD (ozekik 2023)	Inline Turtle in Markdown body	Nonstandard `{}` annotations	Full RDF
MD-LD (davay42 2026)	Inline RDF via `{=iri}` syntax	Nonstandard `{=iri}` annotations	Full RDF
json-ld-markdown (iunera 2025)	Schema.org inference from structure	None (inference-based)	JSON-LD output only
LMD (this paper)	JSON-LD frontmatter, SHACL validation, SPARQL query	Standard JSON-LD/YAML in frontmatter	Full RDF 1.1

DataBooks (Cagle 2026) is the closest intellectual precedent. LMD adopts the DataBooks vision but diverges by requiring JSON-LD frontmatter (not YAML) and zero custom inline syntax, enabling native RDF integration without any transformation step.

Markdown-LD and **MD-LD** both embed RDF triples directly in Markdown body text using custom annotation syntax. These approaches are complementary to LMD: LMD addresses document-level typing and corpus-wide validation, while inline annotation systems address triple-level granularity within a single document. However, their nonstandard syntax can cause unpredictable rendering in standard Markdown renderers, and they require editor plugins or preprocessing.

json-ld-markdown infers Schema.org JSON-LD from plain Markdown structure (headings, tables, lists). This is useful for SEO and consumption-oriented pipelines, but inference necessarily loses information that only author- intent typing can provide.

LMD differs from all prior work in three key ways: 1. Frontmatter is valid JSON-LD by construction, not converted to RDF 2. Validation and query are specified using W3C standards (SHACL, SPARQL) 3. Capabilities are layered (Core, Validation, Query, Publish) and independently adoptable

5 Implementations

LMD has two reference implementations, both available as standard library packages:

5.1 Python Implementation

`linked-markdown` on PyPI (github.com/wazootech/linked-markdown-py)

An `extract()` function that parses LMD documents from a filesystem corpus, resolving `@id` and `@type`, producing a JSON-LD dictionary in `.attrs`. The resulting JSON-LD can be loaded directly into an `rdflib.Graph` for SHACL validation (via `pyshacl`) and SPARQL 1.1 query execution.

Key architectural decisions include: zero required configuration beyond a corpus root; automatic shape and axiom discovery from convention-based directories; and strict separation of parsing, validation, query, and publish into independently invocable CLI subcommands.

5.2 TypeScript Implementation

`@wazoo/linked-markdown` on JSR (github.com/wazootech/linked-markdown-ts)

An `extract()` function returning a JSON-LD document in `.attrs`, compatible with the `jsonld` npm package for RDF/JS quad production, SHACL validation, and SPARQL query execution. Available for Deno, Node, Bun, and browser via CDN.

6 Discussion

A central design choice is that LMD does not define its own vocabulary for general-purpose metadata. The `lmd:` namespace is reserved for protocol-layer concerns: document types, versioning, validation bindings, and provenance. For subject-matter properties, LMD documents use established vocabularies such as Schema.org, Dublin Core, FOAF, and PROV-O. This keeps the `lmd:` namespace small and scoped, and means LMD is not a competing vocabulary standard — it is a Markdown-compatible container in which those vocabularies live.

6.1 Limitations

LMD currently requires JSON-LD knowledge for document creation. While the default context provides commonly used prefixes (`schema`, `lmd`, `rdf`, `rdfs`, `owl`, `sh`, `xsd`, `dc`, `foaf`, `prov`), authors must understand basic RDF concepts (IRIs, types, predicates) to create typed documents. Tooling improvements — such as CLI scaffolding commands (`wiki init`, `wiki new`) — reduce this burden.

Interoperability with non-LMD Markdown tools is a deliberate design constraint. Any CommonMark renderer can display an LMD document, but only LMD processors can validate, query, or publish it. This is acceptable: the protocol follows the robustness principle of being liberal in what it accepts (any Markdown file with JSON-LD frontmatter) and conservative in what it produces (valid RDF).

6.2 Adoption Path

LMD is incrementally adoptable. A single file with an `@id` and `@type` in its frontmatter is a valid LMD document. Adding a shapes file enables validation. Adding a SPARQL query enables query. Adding a publish configuration enables HTML output. Each capability can be adopted independently.

For existing Markdown corpora (Obsidian vaults, GitHub wikis, Jekyll sites), adoption can begin with a single frontmatter field and grow organically.

7 Future Work

- **LMD Schema Registry**: A community-maintained registry of SHACL shapes for common document types (Person, Organization, Article, Book, etc.), analogous to Schema.org but expressed as SHACL.
- **Crawl and Federate**: Cross-corpus SPARQL federation enabling queries across multiple LMD-published sites.
- **lmd:scheme**: A dedicated URI scheme for LMD document resolution, potentially registered with IANA.
- **Interoperability Layer**: Bridge to other Markdown semantic systems (Markdown-LD, MD-LD) via SHACL shapes and RDF transformation pipelines.

8 Conclusion

LMD demonstrates that Markdown can serve as a first-class semantic infrastructure without sacrificing its core virtues: simplicity, portability, and human readability. By restricting protocol semantics to standard JSON-LD frontmatter and CommonMark links, LMD enables typed, validatable, queryable Markdown documents that remain fully compatible with existing tools and workflows.

The full specification follows in Appendix A.

9 References

Cagle, Shannon. 2026. "DataBooks: Markdown as Semantic Infrastructure." https://ontologist.substack.com/p/databooks-markdown-as-semantic-infrastructure.

davay42. 2026. "MD-LD: Inline RDF Annotations in Markdown." https://github.com/davay42/md-ld.

Harris, Steve, and Andy Seaborne. 2013. "SPARQL 1.1 Query Language." W3C Recommendation. W3C. https://www.w3.org/TR/sparql11-query/.

iunera. 2025. "Json-Ld-Markdown: Schema.org Inference from Markdown." https://github.com/iunera/json-ld-markdown.

Knublauch, Holger, and Dimitris Kontokostas. 2017. "Shapes Constraint Language (SHACL)." W3C Recommendation. W3C. https://www.w3.org/TR/shacl/.

Motik, Boris, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2012. "OWL 2 Web Ontology Language Profiles (Second Edition)." W3C Recommendation. W3C. https://www.w3.org/TR/owl2-profiles/.

ozekik. 2023. "Markdown-LD: Literate Programming with Turtle in Markdown." https://github.com/ozekik/markdown-ld.

10 Appendix A: LMD Specification

This appendix contains the complete LMD specification, also available at `https://wazootech.github.io/linked-mark`

11 Linked Markdown (LMD)

Specification Version 0.1.0 — Draft

Repository: github.com/wazootech/linked-markdown **Reference Implementations:** - TypeScript: `@wazoo/linked-markdown` on JSR (github.com/wazootech/linked-markdown-ts) - Python: `linked-markdown` on PyPI (github.com/wazootech/linked-markdown-py) **License:** MIT

11.1 Status of This Document

This document was published by the JSON for Linked Data Community Group. It is a draft Editor's Draft and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to cite this document as other than work in progress.

If you wish to make comments regarding this document, please open an issue or contact the group on their public mailing list public-json-ld@w3.org.

This document is governed by the W3C Community Contributor License Agreement (CLA).

11.2 1. Introduction

Linked Markdown (LMD) is a specification for structuring, validating, and querying typed Markdown documents as first-class semantic graph nodes.

LMD defines a protocol over standard `.md` files. An LMD document is simultaneously:

- **Valid Markdown** — renderable by any CommonMark-compliant renderer, including GitHub, Obsidian, VS Code, and Pandoc.
- **Valid JSON-LD** — frontmatter is a valid JSON-LD node, loadable by any RDF toolchain including `rdflib`, Apache Jena, and Oxigraph.
- **Incrementally adoptable** — a single file with an `@type` field and an `@id` field is an LMD document. The protocol adds capability without breaking existing workflows.

No custom syntax is introduced. No new file extension is required. The protocol lives entirely in the frontmatter, the linking conventions, and the validation shapes applied by conforming processors.

LMD is the Worlds-aligned specification for defining item types, shapes, and orchestrating world memory within the Wazoo Worlds paradigm. It is the formalization of the design pattern described by DataBooks (Cagle, Shannon 2026) with the critical distinctions that LMD frontmatter is valid JSON-LD (not plain YAML) and LMD requires no inline annotation syntax.

11.2.1 1.1. Design Goals

1. **Zero custom syntax.** LMD adds no nonstandard Markdown syntax. All protocol semantics are expressed through standard JSON-LD frontmatter and standard Markdown links.
2. **Valid by default.** A vanilla `.md` file with JSON-LD frontmatter is valid LMD. The protocol does not require any special Markdown dialect.
3. **Layered capability.** A processor may validate, query, link-check, or publish — each capability is independent and optional at the processor level.
4. **Standard RDF foundation.** LMD does not invent a new data model. It maps directly onto RDF 1.1, JSON-LD 1.1, SHACL, OWL-RL, and SPARQL 1.1.
5. **Deterministic structure.** Two conforming processors that validate the same corpus against the same shapes produce the same results.

11.2.2 1.3. Prior Art and Related Work

- **DataBooks (Cagle, Shannon, 2026)** — A design pattern for Markdown as semantic infrastructure. LMD adopts the DataBooks vision but diverges by requiring JSON-LD frontmatter (not YAML) and zero custom inline syntax.
- **Markdown-LD (ozekik, 2023)** — A literate programming approach to embedding Turtle in Markdown body text using inline RDF syntax. Complementary to LMD; LMD addresses document-level typing and validation rather than inline triple annotation.

- **MD-LD (davay42, 2026)** — A zero-dependency JavaScript library for inline RDF annotations in Markdown using `{=iri}` syntax. The nonstandard annotation syntax can cause unpredictable rendering. LMD avoids this entirely by restricting protocol semantics to frontmatter.
- **json-ld-markdown (iunera, 2025)** — A transformer that infers Schema.org JSON-LD from plain Markdown structure (headings, tables, lists). Addresses a different concern (SEO/consumption-oriented inference vs. LMD's author-intent typing).

11.2.3 1.4. Protocol Status and Versioning

This specification uses semantic versioning. Versions before 1.0.0 are drafts and may change incompatibly between minor versions. Once 1.0.0 is published, breaking changes require a major version bump.

11.3 2. Conformance

11.3.1 2.1. Document Conformance

A document conforms to LMD if and only if:

1. It is a syntactically valid Markdown document per the CommonMark specification.
2. It contains exactly one JSON-LD frontmatter block delimited by `---` at the start of the file.
3. The frontmatter block is valid JSON-LD per JSON-LD 1.1.

The minimum viable LMD document:

```
---
---
---

# My Item
Content here.
```

A processor MUST NOT reject a conforming LMD document for lacking optional fields such as `@context`, `name`, or `description`.

11.3.2 2.2. Processor Conformance

A processor conforms to LMD if it implements at least one of the following capability tiers at the required conformance level:

- **LMD-Core** — Must parse frontmatter as JSON-LD, resolve `@id` and `@type`, and produce an RDF 1.1 graph.
- **LMD-Validation** — Must implement SHACL validation per SHACL 1.1.
- **LMD-Query** — Must implement SPARQL 1.1 query execution against the LMD graph.
- **LMD-Publish** — Must produce static HTML output consistent with the LMD document's semantic content.

Tier identifiers are used for capability discovery. A processor may advertise: `Conforms-To: LMD-Core, LMD-Validation`.

11.3.3 2.3. Shape Conformance

A SHACL shapes graph conforms to LMD if it is valid SHACL per the W3C SHACL specification and targets at least one LMD document type via `sh:targetClass`.

11.4 3. The LMD Document Model

11.4.1 3.1. Document Identity

An LMD document SHOULD declare a canonical IRI in its `@id` field. When present, this IRI is the document's identity within the LMD corpus and serves as the RDF subject for all triples generated from the document's frontmatter.

The `@id` SHOULD be a dereferenceable HTTP(S) IRI. The `@id` MAY be a URN or tag URI for documents that are not publicly hosted.

```
"@id": https://example.org/docs/people/alice-smith
```

11.4.2 3.2. Document Type

An LMD document SHOULD declare its semantic type via `@type`. The value SHOULD be one or more IRI references, which may use CURIE notation when a `@context` is present.

```
@type:
  - schema:Person
  - lmd:Document
```

11.4.3 3.3. The JSON-LD Context

The `@context` field defines the prefix mappings for CURIE expansion within the frontmatter. A processor MUST resolve all CURIEs in the frontmatter against the active context before producing RDF.

The context MUST include at minimum:

```
@context:
  schema: https://schema.org/
  lmd: https://wazootech.github.io/linked-markdown/ns#
```

Processors SHOULD provide a default context that includes commonly used prefixes (`schema`, `lmd`, `rdf`, `rdfs`, `owl`, `sh`, `xsd`, `dc`, `dcterms`, `foaf`, `prov`). A document may override any default prefix.

11.4.4 3.4. Vocabulary Conventions

The `lmd:` prefix defines terms specific to the LMD protocol layer — document types, versioning, validation metadata, and provenance stamps. These terms describe a document's relationship to the LMD protocol rather than its subject matter.

For subject-matter and general-purpose metadata, documents SHOULD use established vocabularies such as Schema.org (`schema:`), Dublin Core (`dcterms:`), FOAF (`foaf:`), or PROV-O (`prov:`). A document that uses only standard vocabularies without any `lmd:`-prefixed properties is a valid LMD document. LMD does not replace existing vocabularies; it provides a Markdown-compatible substrate in which they coexist.

A processor MAY define equivalence mappings between `lmd:` terms and terms in other vocabularies to improve interoperability with non-LMD RDF consumers. Such mappings are processor-specific and not required for conformance.

11.4.5 3.5. Body Content

The Markdown body text (everything after the frontmatter) is part of the LMD document and is addressable as an RDF literal via the configured content predicate. The default content predicate is `schema:articleBody`.

A processor MAY include the body text as a literal triple in the document's RDF graph:

```
<id> schema:articleBody "Body text here..." .
```

A processor MUST NOT require body content. An LMD document may consist only of frontmatter.

11.4.6 3.6. Links as Properties

Standard Markdown links `[text](target)` in the body are interpreted as potential RDF object properties by a processor, but the protocol does not mandate automatic triple generation from arbitrary inline links. Only links whose semantic role is declared (via frontmatter, shape, or explicit convention such as `wikilinks` to other LMD documents) produce triples.

11.5 4. Frontmatter as JSON-LD

11.5.1 4.1. Syntax

Frontmatter MUST be a JSON-LD 1.1 document delimited by `---`. The frontmatter may use either JSON or YAML syntax, as YAML is a superset of JSON.

```
---
{
  "@id": "https://example.org/doc",
  "@type": "schema:Article",
  "@context": {
    "schema": "https://schema.org/"
  },
  "schema:name": "Example Document"
}
---
```

YAML syntax is also conforming:

```
---
"@id": https://example.org/doc
"@type": schema:Article
@context:
  schema: https://schema.org/
name: Example Document
---
```

11.5.2 4.2. Required Fields

Field	Type	Description
`@id`	IRI	Canonical identifier for the document
`@type`	IRI or IRI[]	RDF type(s) of the document

11.5.3 4.3. Recommended Fields

Field	Type	Description
`@context`	Object	CURIE prefix mappings
`name`	string	Human-readable title
`description`	string	Short summary (aim for 200 chars or less)

A processor MUST NOT reject a conforming LMD document for lacking recommended fields.

11.5.4 4.4. CURIE Resolution

When a `@context` is present, all CURIEs (e.g., `schema:Person`) in the frontmatter are expanded to full IRIs using the context's prefix map. A processor MUST reject a document containing unresolvable CURIEs when no matching prefix is defined.

11.5.5 4.5. Relation to RDF

Each frontmatter field that is not a JSON-LD keyword (`@id`, `@type`, `@context`) is mapped to an RDF predicate-value pair with the document's `@id` as the subject. Arrays become multiple triples with the same subject-predicate. Nested objects (when supported by the processor) become blank nodes or named nodes per JSON-LD 1.1 framing rules.

11.6 5. Document Linking

11.6.1 5.1. Intra-Corpus Links

Links between LMD documents in the same corpus are expressed through Markdown links whose target is another LMD document's filename or IRI. When the target document exists in the corpus, the link is resolved to that document's `@id` IRI.

```
See [Alice Smith](Alice_Smith.md) for details.
```

A processor SHOULD verify that linked targets exist within the corpus and SHOULD report broken links as warnings or errors depending on processor configuration.

11.6.2 5.2. External Links

Links to IRIs outside the LMD corpus are treated as external references. A processor MAY resolve them but MUST NOT require resolution. External links are preserved as RDF object properties when the predicate is typed.

11.6.3 5.3. Fragment Identifiers

An LMD document may contain sections addressed by fragment identifier (e.g., `#section-2`). These fragments MAY be typed as `rdf:HTML` or `rdf:XMLLiteral` content within the document's RDF representation.

11.7 6. Validation

11.7.1 6.1. SHACL Validation

An LMD processor that supports validation MUST apply SHACL shapes loaded from the corpus's shapes directory against every LMD document. Validation follows the SHACL 1.1 specification:

- Compliance is determined by `sh:targetClass` matching the document's `@type`.
- Each shape defines property constraints (`sh:path`, `sh:datatype`, `sh:minCount`, `sh:maxCount`, `sh:pattern`, etc.).
- A shape may reference other shapes via `sh:node`.
- A shape may reference JSON Schema via `lmd:jsonSchema` (see 6.2).

11.7.2 6.2. JSON Schema Integration

An LMD shape MAY declare a JSON Schema binding via the `lmd:jsonSchema` property. When present, a processor MUST validate the frontmatter's JSON representation against the referenced JSON Schema. This enables validation of deeply nested structures that are awkward to express in SHACL alone.

```
@prefix lmd: <https://wazootech.github.io/linked-markdown/ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
```

```
wiki:ContactShape a sh:NodeShape ;
    sh:targetClass wiki:Contact ;
    lmd:jsonSchema "contact.schema.json" .
```

11.7.3 6.3. Shape Location

Shapes are loaded from a directory declared in the corpus configuration. The default location is `shapes/` relative to the corpus root. Shapes may be in Turtle (`.ttl`), JSON-LD (`.jsonld`), or RDF/XML (`.rdf`) format.

11.8 7. Inference

11.8.1 7.1. OWL-RL Deductive Reasoning

An LMD processor that supports inference SHOULD apply OWL-RL deductive reasoning to expand the LMD corpus graph. The OWL-RL rule set is defined by OWL 2 RL.

Inference enables subclass reasoning, property chain expansion, and domain/range inference:

```
# document.md
---
"@id": wiki:alice-smith
@type: wiki:Engineer
---
```

With the axiom `wiki:Engineer rdfs:subClassOf schema:Person`, a reasoning processor infers `wiki:alice-smith a schema:Person`.

11.8.2 7.2. Custom Axioms

A processor MAY load custom axioms from the corpus. Axioms are Turtle or JSON-LD files that declare OWL class hierarchies, property chains, or SWRL-like rules. Axiom files live in the `axioms/` directory by default.

11.8.3 7.3. Opt-Out

A processor MUST allow clients to opt out of inference. The default inference mode is processor-specific.

11.9 8. Query

11.9.1 8.1. SPARQL 1.1

An LMD processor that supports query MUST implement SPARQL 1.1 Query Language and Protocol. Processors SHOULD support SELECT, CONSTRUCT, ASK, and DESCRIBE query forms.

11.9.2 8.2. Embedded Query Blocks

An LMD document may contain SPARQL query blocks embedded as fenced code blocks. These blocks are marked with the language tag `sparql`:

````
```sparql
SELECT ?name ?email WHERE {
  ?person a schema:Person ;
          schema:name ?name ;
          schema:email ?email .
}
```
````

A processor MAY render the results of embedded queries inline (below the query block) when generating output. The processor MUST indicate where results begin and end with processor-specific comments to enable round-trip updates.

11.9.3 8.3. Result Formats

Query results SHOULD be representable in the following formats, at minimum: JSON, CSV, TSV, Markdown table, and HTML table. A processor determines the default result format.

11.10 9. Serialization

11.10.1 9.1. RDF Export

An LMD processor MUST be capable of exporting the LMD corpus graph in at least one of the following RDF serialization formats:

- JSON-LD 1.1 (compacted or expanded)
- Turtle (`.ttl`)
- RDF/XML (`.rdf`)

Processors SHOULD also support N-Triples, TriG, and N-Quads.

11.10.2 9.2. JSON-LD Export Specifics

When exporting as JSON-LD, a processor MUST preserve the document's `@context` in the output. The export MUST include all triples derived from the corpus, including those produced by inference (if inference was applied).

11.11 10. Publishing

11.11.1 10.1. Static HTML

An LMD processor that supports publishing SHOULD produce a static HTML site from the LMD corpus. The output MUST include:

- A page for each LMD document, with human-readable rendering of its content.
- Navigation between linked documents.
- A machine-readable representation of each document (JSON-LD, Turtle, etc.) linked from the HTML page.

11.11.2 10.2. URL Structure

Page URLs SHOULD be derived from the document's filename, minus the `.md` extension. A processor MAY support alternative URL styles (directory-style `/slug/` vs. file-style `/slug.html`).

11.11.3 10.3. Content Negotiation

A processor MAY support HTTP content negotiation, serving HTML to browsers and machine-readable formats (JSON-LD, Turtle) to agents. The `Accept: text/markdown` header SHOULD return the raw Markdown source, following the precedent established by Cloudflare Markdown for Agents.

11.12 11. Provenance

11.12.1 11.1. Process Stamps

LMD documents may include provenance metadata in their frontmatter, modeled after the PROV-O ontology. A process stamp records how a document was produced:

```
lmd:provenance:
  lmd:transformer: lmd:cli
  lmd:inputs:
    - https://example.org/docs/source-doc
  lmd:timestamp: "2026-06-23T12:00:00Z"
  lmd:agent:
    @type: schema:Person
    schema:name: "Alice Smith"
```

11.12.2 11.2. PROV-O Alignment

Process stamps SHOULD align with the W3C PROV-O ontology: - `lmd:provenance` maps to `prov:Activity` - `lmd:inputs` maps to `prov:used` - `lmd:agent` maps to `prov:wasAssociatedWith` - The document itself is a `prov:Entity` generated by the activity

11.13 12. Security Considerations

11.13.1 12.1. IRI Injection

IRIs in `@id` and link targets MUST be validated to prevent injection of unexpected schemes (e.g., `javascript:`, `data:`). A processor SHOULD reject IRIs with non-http schemes in publishing and query contexts.

11.13.2 12.2. Schema Loading

When loading shapes or axioms from remote IRIs, a processor SHOULD validate the remote content's media type and MAY refuse to load from untrusted sources.

11.13.3 12.3. SPARQL Injection

Embedded SPARQL blocks MUST be treated as untrusted input when loaded from untrusted corpora. A processor SHOULD apply read-only execution mode for embedded queries and MUST prevent CONSTRUCT or UPDATE operations that would modify the corpus.

11.14 13. IANA Considerations

Linked Markdown does not currently require any IANA registrations. Future versions may request: - A media type registration (e.g., `text/lmd` or `application/lmd+json`) - A URI scheme registration (if a dedicated `lmd:` scheme is desired)

These considerations are deferred until the protocol reaches stability at version 1.0.0.

11.15 Appendix A: Complete LMD Document Example

```
---
"@id": https://example.org/docs/people/alice-smith
@type:
  - schema:Person
  - lmd:Document
@context:
  schema: https://schema.org/
  lmd: https://wazootech.github.io/linked-markdown/ns#
  wiki: https://example.org/docs/
name: Alice Smith
description: Profile page for Alice Smith, software engineer.
```

```
schema:givenName: Alice
schema:familyName: Smith
schema:email: alice@example.com
schema:knows:
  - wiki:bob-jones
  - wiki:carol-davis
schema:jobTitle: Senior Software Engineer
lmd:provenance:
  lmd:transformer: lmd:cli
  lmd:timestamp: "2026-06-23T10:00:00Z"
  lmd:agent:
    @type: schema:Person
    schema:name: Alice Smith
---

# Alice Smith

Alice is a senior software engineer at Example Corp.

## Biography

Alice has been building semantic web applications since 2020.

## Related

- See [Bob Jones](bob-jones.md) for a colleague
- See [Carol Davis](carol-davis.md) for another colleague
```

11.16 Appendix B: LMD Namespace

The `lmd:` prefix expands to `https://wazootech.github.io/linked-markdown/ns#`. The following terms are defined:

Term	Description
`lmd:Document`	The base type for all LMD documents
`lmd:Specification`	The type for the LMD specification document
`lmd:version`	The LMD specification version a document targets
`lmd:status`	The document's protocol status (Draft, Stable, Deprecated)
`lmd:published`	The document's publication date
`lmd:license`	The document's license IRI
`lmd:repository`	The document's canonical repository
`lmd:supersedes`	An IRI this specification replaces
`lmd:jsonSchema`	Links a SHACL shape to an external JSON Schema file
`lmd:provenance`	Provenance / process stamp metadata
`lmd:transformer`	The tool or agent that produced the document
`lmd:inputs`	Input documents used to produce the document
`lmd:timestamp`	The production timestamp
`lmd:agent`	The agent responsible for production

11.17 Appendix C: Glossary

Term	Definition
LMD	Linked Markdown
Corpus	A collection of LMD documents sharing a configuration
Document	A single `.md` file with LMD-conforming frontmatter
Shape	A SHACL constraint definition for validating document structure
Axiom	A set of OWL or SWRL rules for deductive reasoning
Processor	Any tool or library implementing one or more LMD tiers
Frontmatter	The JSON-LD metadata block at the start of an LMD document

---

11.18 Appendix D: References

- CommonMark Spec — Standard Markdown syntax
- JSON-LD 1.1 — JSON-based RDF serialization
- RDF 1.1 — RDF data model
- SHACL 1.1 — Shapes Constraint Language
- OWL 2 RL — OWL 2 RL profile
- SPARQL 1.1 — SPARQL query language
- PROV-O — Provenance ontology
- RFC 4151 — The 'tag' URI scheme
- DataBooks: Markdown as Semantic Infrastructure — Cagle, Shannon 2026

Repository: github.com/wazootech/linked-markdown-paper Specification: github.com/wazootech/linked-markdown TypeScript Implementation: github.com/wazootech/linked-markdown-ts Python Implementation: github.com/wazootech/linked-markdown-py