

Linked Data Event Streams

Living Standard, 3 July 2025

This version:

<https://w3id.org/ldes/specification>


Issue Tracking:

[GitHub](#)

[Inline In Spec](#)

Editor:

[Pieter Colpaert](#)

 To the extent possible under law, the editors have waived all copyright and related or neighboring rights to this work. In addition, as of 3 July 2025, the editors have made this specification available under the [Open Web Foundation Agreement Version 1.0](https://www.openwebfoundation.org/the-agreements/the-owf-1-0-agreements-granted-claims/owfa-1-0), which is available at <https://www.openwebfoundation.org/the-agreements/the-owf-1-0-agreements-granted-claims/owfa-1-0>. Parts of this work may be from another specification document. If so, those parts are instead covered by the license of that specification document.

Abstract

A Linked Data Event Stream (LDES) is an append-only collection of members described using the Resource Description Framework (RDF). The specification explains how a client can replicate the history of an event stream, and how it can then remain synchronized as new members are published.

Table of Contents

1	Introduction
2	Overview
3	Context information
3.1	Versions and transactions
3.2	Retention policies
4	Vocabulary
4.1	ldes:EventStream
4.2	ldes:timestampPath
4.3	ldes:sequencePath
4.4	ldes:versionOfPath
4.5	ldes:versionTimestampPath
4.6	ldes:versionSequencePath
4.7	ldes:EventSource
4.8	ldes:retentionPolicy
4.9	ldes:RetentionPolicy
4.9.1	ldes:startingFrom

- 4.9.2 ldes:versionDuration
- 4.9.3 ldes:versionAmount
- 4.9.4 ldes:versionDeleteDuration
- 4.9.5 ldes:fullLogDuration
- 4.9.6 Former retention policies terms
- 4.10 Terms for versioning and transactions on top of ldes:EventStream
 - 4.10.1 ldes:versionCreatePath
 - 4.10.2 ldes:versionUpdatePath
 - 4.10.3 ldes:versionDeletePath
 - 4.10.4 ldes:versionCreateObject
 - 4.10.5 ldes:versionUpdateObject
 - 4.10.6 ldes:versionDeleteObject
 - 4.10.7 ldes:transactionPath
 - 4.10.8 ldes:transactionFinalizedPath
 - 4.10.9 ldes:transactionFinalizedObject

Conformance

References

Normative References

Issues Index

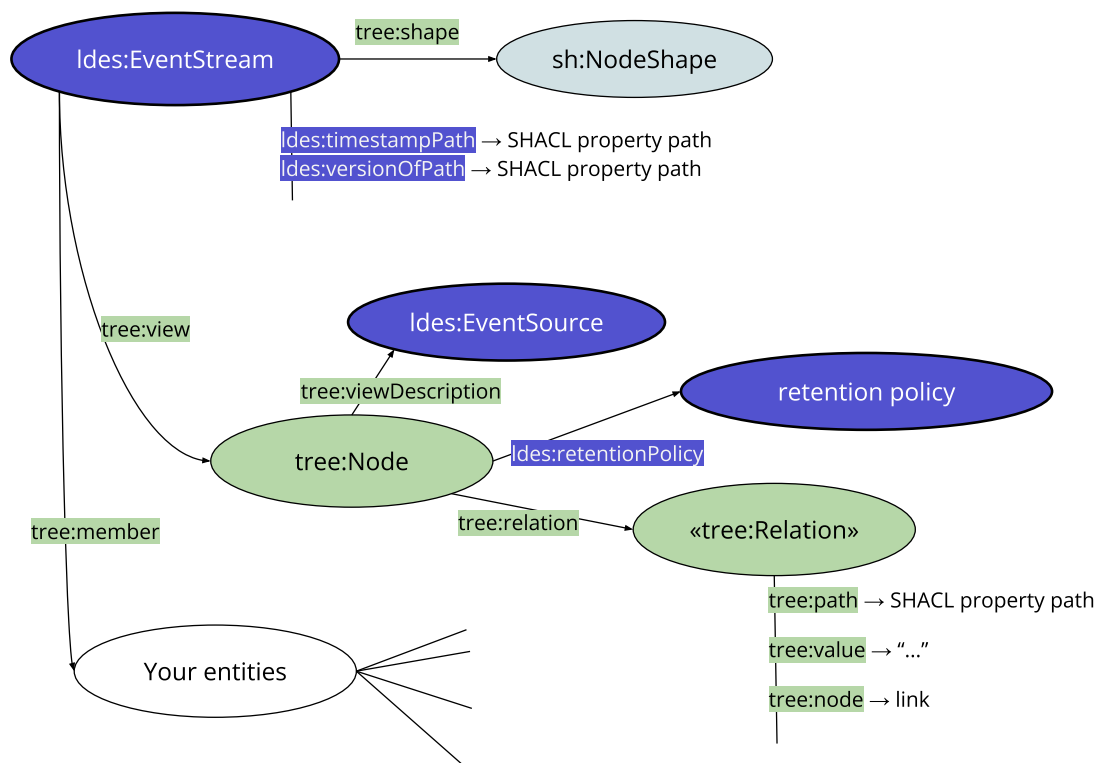
§ 1. Introduction

An **LDES client** is a piece of software used by a *consumer* that accepts the URL to an entry point, and returns a stream of members of the corresponding `ldes:EventStream`. The data stream emits the history that is available from this entry point, and once the consumer has caught up with the stream, it remains synchronized as new members are published.

The client does this by extending upon a subset of [the W3C TREE hypermedia specification](#). This *extension* introduces specialized terms for dealing with append-only collections, referred to as *event streams*. For example, one can indicate what time-based property in the member is used for indicating the order of the event stream, indicate the retention policy as a promise from the producer to the consumer, or detail how to deal with version-based members.

ISSUE 1 More extensions should be specified w.r.t. [HTTP status codes](#), or [keeping state](#). This should be further detailed in a chapter after the overview. ¹

§ 2. Overview



A Linked Data Event Stream (LDES) (`ldes:EventStream`) is a collection (`rdfs:subClassOf tree:Collection`) of members that cannot be updated or removed once they are published, with each member being a set of RDF quads ([\[rdf-primer\]](#)). This way, the collection of members becomes an append-only log or *event stream*.

Following the [TREE specification](#), this event stream is published using one or more HTTP resources. When more resources are used, these pages, or `tree:Nodes`, will be structured according to a search tree. Therefore we will use the terms *root node* for the first page, and *subsequent node* for every next page in the structure.

In the root node, the client will expect these properties to be described on the `ldes:EventStream` entity:

- `ldes:timestampPath`: this is a [SHACL property path](#) (all further `*Path` properties are as well) that sets the chronological time with a `xsd:dateTime` literal within each member. This timestamp determines the chronological order in which members of the event stream are added. When `ldes:timestampPath` is set, no member can be added to the LDES with a timestamp earlier than the latest published member.
- `ldes:sequencePath`: when the LDES producer wants to make clear what the ordering is within members with the same timestamp for the `ldes:timestampPath`, this property defines, based on the [\[xpath-functions-31\] comparison operator](#), what `xsd`-literals define the order of processing. When no `ldes:timestampPath` has been set, the `ldes:sequencePath` defines the sequence for all members in the LDES.
- `ldes:versionOfPath`: when your entities are versioned, this property points at the object that tells you what entity it is a version of (e.g., `dcterms:isVersionOf`).
- `ldes:versionTimestampPath`: when those versions are not published chronologically, LDES providers have the possibility to indicate a different version timestamp to establish the latest version.
- `ldes:versionSequencePath`: when the versions do not follow the order in `ldes:timestampPath` and `ldes:sequencePath`, and for when `ldes:versionTimestampPath` are the same for multiple members, or for when this property is not set. E.g., for out of order publishing of `v0.1` → `v0.2`, in which `v0.2` may have been published by the server before `v0.1`.

- `tree:shape`: a [\[SHACL\]](#) shape that can be used to select a search tree in the discovery phase, as well as to validate the members in the event stream. The `sh:NodeShape` linked validates each target of the `tree:member` property. A validator in a consumer pipeline MUST ignore other targets.
- `tree:view`: connects the collection to the current page, or points to one specific root node after dereferencing the `ldes:EventStream` identifier.

In the root node, the current node identified by the URL of the page (a provider can achieve this simply by using a relative IRI <>) will be further described using these properties:

- `ldes:retentionPolicy`: indicates a retention policy (see next the dedicated chapter [\[\]\(#retention\)](#)).
- `tree:viewDescription`: can as well contain the retention policy, or other context data about this view of the LDES (e.g., the `dcat:Distribution`, the `tree:SearchTree`, or the `ldes:EventSource`) as a named entity. This is useful for example if a producer would like to disambiguate the IRI for the `ldes:EventSource` from the root `tree:Node`. By default, the `tree:viewDescription` points at the root node.

The client MUST implement the [initialization of the TREE specification](#), and they MAY extract the `tree:search` form.

In any `tree:Node` – root node or subsequent node – the client expects to find 0 or more members of the `ldes:EventStream` using the `tree:member` property. The subject is the event stream instance and the object is the root focus node of a member. An LDES client MUST implement the [Member Extraction Algorithm of the TREE specification](#) to retrieve the full set of quads of the member. In an `ldes:EventStream`, the object of the `tree:member` triple can only be an IRI as this IRI will be used in the state to check whether the member has already been emitted or not.

EXAMPLE 1

An example record from a sensor observation dataset in the [\[turtle\]](#) format:

```
ex:Observations a ldes:EventStream ;
    ldes:timestampPath sosa:resultTime ;
    tree:shape ex:shape1.shacl ;
    tree:view <> ;
    tree:member ex:Observation1 .

ex:Observation1 a sosa:Observation ;
    sosa:resultTime "2026-01-01T00:00:00Z"^^xsd:dateTime ;
    sosa:hasSimpleResult "..."
```

EXAMPLE 2

An example record from a base registry of addresses in the [\[trig\]](#) format:

```
ex:AddressRecords a ldes:EventStream ;
    ldes:timestampPath dcterms:created ;
    ldes:versionOfPath dcterms:isVersionOf ;
    tree:shape ex:shape2.shacl ;
    tree:view <> ;
    tree:member ex:AddressRecord1-activity1 .

ex:AddressRecord1-activity1 dcterms:created "2026-01-01T00:00:00Z"^^xsd:dateTime ;
    adms:versionNotes "First version of this address" ;
    dcterms:isVersionOf ex:AddressRecord1 .

ex:AddressRecord1-activity1 {
    ex:AddressRecord1 dcterms:title "Streetname X, ZIP Municipality, Country" .
}
```

In any `tree:Node`, root node or subsequent node, the client expects to find zero or more `tree:relation` properties, containing a description of the `tree:Relations` from this node to subsequent nodes. A client **MUST** traverse the relations cfr. the TREE chapters on [traversing the search tree](#). A client **MUST** keep its own *state* to know when to refetch certain `tree:Nodes`.

ISSUE 2 We should refer here to a new next chapter on how to gracefully iterate over the pages and how to keep the state in more detail cfr. the extensions in Issue 1. We can then also indicate that a client **MAY** implement the text on [pruning branches](#) related to interpreting comparators for `xsd:dateTime` literals if it wants to detect immutable pages via the `timestampPath`.

ISSUE 3 More specific server documentation should be found in a Server Primer (to do), such as containing a [link to the JSON-LD context](#), [official SHACL shapes for LDES](#) to validate your pages, best practices for publishing an LDES for reaching an optimal performance, best practices for enveloping your data using named graphs, how to build a status log for the use case of an aggregator or harvester, etc.

§ 3. Context information

A client **MUST** have a way to make available context information to processors further down the pipeline set up by a consumer. A consumer will be able to make decisions on how to further process the event stream using this context information.

§ 3.1. Versions and transactions

Consumers can use the LDES version properties to decide what action to take. E.g., when the consumer understands the members are versioned, it can upsert the members on each update. If it understands something was created instead of updated, it can just add it into the store without removing statements first, and when a deletion comes in, it knows it can remove the statements associated with the previous insert or upsert. To that extent, on the `ldes:EventStream` entity, these properties can be used, which are further explained in the vocabulary.

- [ldes:versionOfPath](#) - such as `dcterms:isVersionOf` or `as:object`

- [ldes:versionDeleteObject](#) - such as `as:Delete`
- [ldes:versionCreateObject](#) - such as `as:Create`
- [ldes:versionUpdateObject](#) - such as `as:Update`
- [ldes:versionDeletePath](#) - defaults to `rdf:type`
- [ldes:versionCreatePath](#) - defaults to `rdf:type`
- [ldes:versionUpdatePath](#) - defaults to `rdf:type`

A consumer can also understand how to process the event stream in a way that the resulting knowledge graph will be consistent, by interpreting transactions using these properties:

- [ldes:transactionPath](#) - points at an identifier for the transaction. The result of evaluating the path can be a literal or a IRI.
- [ldes:transactionFinalizedPath](#) - points at the object
- [ldes:transactionFinalizedObject](#) - the value that the object needs to be in order to be finalized. Defaults to the `"true"^^xsd:boolean`.

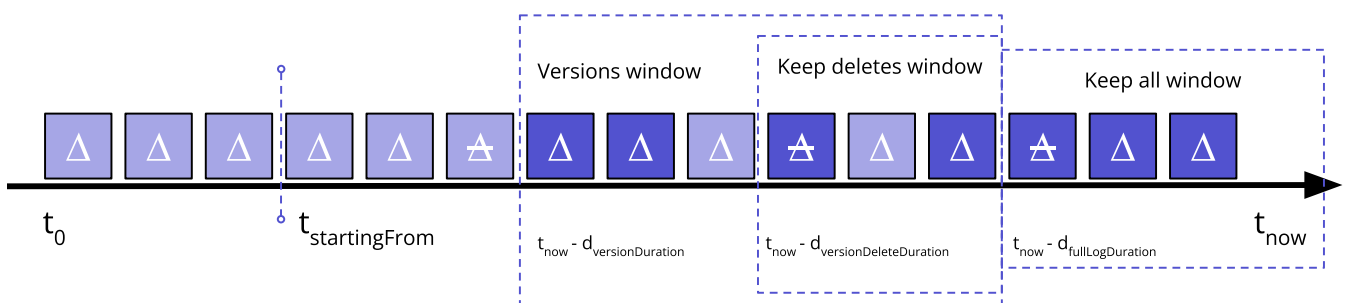
If one of the processors in a pipeline relies on transactions, the LDES client MUST force to emit the members in chronological order according to the `ldes:timestampPath` and `ldes:sequencePath`. When processing multiple members of the transaction with the same `ldes:timestampPath`, the client MUST ensure the member that finalizes the transaction is emitted last.

When the IRI in the object of the `tree:member` triple is also used as a named graph, an LDES consumer MAY assume the payload of the upsert is in the named graph. A consumer MUST implement a way to find back this group of triples in case an update or deletion comes in.

§ 3.2. Retention policies

The goal of a retention policy is to indicate in what way a specific view will not be able to provide a complete history of the event stream to the consumer. This can help a consumer in the discovery phase to pick an specific LDES view, or help the consumer to detect non-viable synchronization set-ups.

When no retention policy is provided in the root node, the consumer MUST assume that all members, that have once been added to the `ldes:EventStream`, are still available from this root node. When a retention policy is provided however, a consumer MUST assume it will not be able to find members outside of the retention policy.



EXAMPLE 3

An example retention policy combining different features from the overview.

```
ex:LDES a ldes:EventStream ;
  ldes:timestampPath as:updated ;
  ldes:transactionPath ex:transactionId ;
  ldes:transactionFinalizedPath ex:transactionEnded ;
  ldes:transactionFinalizedObject true ;
  ldes:versionOfPath as:object ;
  ldes:versionDeleteObject as:Delete ;
  ldes:versionCreateObject as:Create ;
  ldes:versionUpdateObject as:Update ;
  tree:view <> .

<> a ldes:EventSource ;
  ldes:retentionPolicy [
    ldes:fullLogDuration "P1Y"^^xsd:duration ;
    ldes:versionAmount 1 ;
    ldes:versionDeleteDuration "P1Y"^^xsd:duration ;
  ] .
```

A retention policy will be described on the root node. The root node itself can contain this information using the property `ldes:retentionPolicy`, or the root node can refer through the property `tree:viewDescription` to an entity on which the retention policy is described using the property `ldes:retentionPolicy`. When the client is processing the root node, it MUST look for a retention policy in both ways.

In the example above, the retention policy has been set on the root node (double typed as the `ldes:EventSource`). When the `ldes:retentionPolicy` would refer to an entity without further statements in the current page, the client MUST assume this view keeps no members at all. Multiple properties can then be added to make the scope of members that are kept larger:

- **`ldes:startingFrom`**: this view only retains members starting from this `xsd:dateTime` with timezone. In combination with other retention policies, this property only enforces the period before the timestamp for which the view will not retain any member.
- **`ldes:fullLogDuration`**: the duration from current time from which all members are retained. Only in combination with `ldes:startingFrom`, and when the `ldes:startingFrom` timestamp is within this window, not all members within the member are retained. No other properties can influence this property.
- **`ldes:versionAmount`**: the amount of versions to keep.
- **`ldes:versionDuration`**: the duration from current time from which an amount of version are kept, to be used together with `ldes:versionAmount`. Defaults the duration of the full event stream.
- **`ldes:versionDeleteDuration`**: the period of time from current time the deletions in the event stream are retained. Before this period, deletions are not retained, regardless of `ldes:versionAmount` or `ldes:versionDuration`.

When using the current time in calculations, the consumer MUST take into account a safe buffer to mitigate clock inaccuracies. The `ldes:timestampPath` points at the timestamp in the member that can be compared with the current time minus the durations. When the `ldes:versionTimestampPath` has been set, the two version durations are to be compared with this timestamp.

Historically, there are more specific type of retention policies that **MUST** remain supported although their use is discouraged in favor of the just introduced retention policy design. These retention policies types are:

1. `ldes:DurationAgoPolicy`: a time-based retention policy in which data generated before a specified duration is not retained.
2. `ldes:LatestVersionSubset`: a version subset based on the latest versions of an entity in the stream.
3. `ldes:PointInTimePolicy`: a point-in-time retention policy in which data generated before a specific time is not retained.

A `ldes:LatestVersionSubset` uses the property `ldes:amount` with as range an `xsd:integer` datatype, indicating the number of versions to keep. By default, this value is set to 1. A `ldes:PointInTimePolicy` uses a `ldes:pointInTime` with an `xsd:dateTime`-typed literal to indicate the point in time on or after which data is kept when compared to a member's timestamp.

§ 4. Vocabulary

Next to re-using terms from the `tree:` vocabulary, the `ldes:` namespace introduced in this document provides a couple of new terms. The base IRI for LDES is <https://w3id.org/ldes#>, and the preferred prefix is `ldes:`. There is a Turtle version available at <https://w3id.org/ldes#Vocabulary>.

§ 4.1. `ldes:EventStream`

The class `ldes:EventStream` is a subclass of `tree:Collection`. The specialization being that all members are immutable, and thus that this `tree:Collection` is append-only.

§ 4.2. `ldes:timestampPath`

The path to the `xsd:dateTime` literal in each member that defines the order of the event stream.

Domain: `ldes:EventStream`

Range: a [SHACL property path](#)

§ 4.3. `ldes:sequencePath`

The path to an `xsd` literal in each member that defines the order of the event stream in addition to the `ldes:timestampPath`.

Domain: `ldes:EventStream`

Range: a [SHACL property path](#)

§ 4.4. `ldes:versionOfPath`

The path to the IRI in each member that defines the entity of which this member is a version.

Domain: `ldes:EventStream`

Range: a [SHACL property path](#)

§ 4.5. `ldes:versionTimestampPath`

For out of order event streams, this defines the path to the `xsd:dateTime` literal in each member that defines the order of versioned members.

Only relevant when the `ldes:versionOfPath` has been set.

Domain: `ldes:EventStream`

Range: a [SHACL property path](#)

§ 4.6. `ldes:versionSequencePath`

For out of order event streams, this defines the path to an `xsd` literal in each member that defines the order of the event stream in addition to the `ldes:versionTimestampPath`.

Domain: `ldes:EventStream`

Range: a [SHACL property path](#)

§ 4.7. `ldes:EventSource`

The class `ldes:EventSource` is a subclass of `dcate:Distribution`, the specialization being that this is a feed that uses a chronological search tree to make available a Linked Data Event Stream in order.

An `ldes:EventSource` can *only* be published on LDEs that have a `ldes:timestampPath` set, and thus will publish their entities in this chronological order.

§ 4.8. `ldes:retentionPolicy`

Links to a retention policy.

Domain: Preferably the root node. Alternatively it can occur on any type of entity that is linked from the root node using `tree:viewDescription`.

Range: `ldes:RetentionPolicy`

§ 4.9. `ldes:RetentionPolicy`

The class for a retention policy that indicates how long members are preserved in this search tree.

§ 4.9.1. `ldes:startingFrom`

The search tree only keeps members starting a certain timestamp.

Domain: `ldes:RetentionPolicy`

Range: `xsd:dateTime` with a timezone

§ 4.9.2. `ldes:versionDuration`

The search tree only keeps its versions, for which an `ldes:versionAmount` MUST have been set, only during a specific window.

Domain: `ldes:RetentionPolicy`

Range: `xsd:duration`

§ 4.9.3. `ldes:versionAmount`

The number of versions to keep. This MUST be a number greater than 0.

Domain: `ldes:RetentionPolicy`

Range: `xsd:integer > 0`

§ 4.9.4. `ldes:versionDeleteDuration`

The search tree only keeps its deletions for a certain duration.

Domain: `ldes:RetentionPolicy`

Range: `xsd:duration`

§ 4.9.5. `ldes:fullLogDuration`

The search tree keeps its full log for a certain duration.

Domain: `ldes:RetentionPolicy`

Range: `xsd:duration`

§ 4.9.6. Former retention policies terms

- `ldes:DurationAgoPolicy`: A retention policy class that uses an `xsd:duration` literal to document a sliding window of data.
- `ldes:LatestVersionSubset`: A retention policy class that select an amount of versions based on the `versionOfPath`.
- `ldes:amount`: The number of versions to keep. This MUST be a number greater than 0.
 - Domain: `ldes:LatestVersionSubset`
 - Range: `xsd:integer`
- `ldes:PointInTimePolicy`: A retention policy class that indicates members are kept starting on a certain point in time.

- `ldes:pointInTime`: The point in time from which members will be available starting from this root node. - Domain: `ldes:PointInTimePolicy` - Range: `xsd:dateTime` including an explicit timezone

§ 4.10. Terms for versioning and transactions on top of `ldes:EventStream`

§ 4.10.1. `ldes:versionCreatePath`

Path indicating where you can do an object check on whether the member represents a create. Defaults to `rdf:type`.

§ 4.10.2. `ldes:versionUpdatePath`

Path indicating where you can do an object check on whether the member represents an update. Defaults to `rdf:type`.

§ 4.10.3. `ldes:versionDeletePath`

Path indicating where you can do an object check on whether the member represents a delete. Defaults to `rdf:type`.

§ 4.10.4. `ldes:versionCreateObject`

If the RDF object matches the object in the version create path, the member represents a create.

§ 4.10.5. `ldes:versionUpdateObject`

If the RDF object matches the object in the version update path, the member represents an update.

§ 4.10.6. `ldes:versionDeleteObject`

If the RDF object matches the object in the version delete path, the member represents a delete.

§ 4.10.7. `ldes:transactionPath`

Path indicating how a member indicates whether it is part of a transaction.

§ 4.10.8. `ldes:transactionFinalizedPath`

Path indicating whether the transaction has been finalized.

§ 4.10.9. `ldes:transactionFinalizedObject`

If the RDF object matches the object in the transaction finalized path, the member indicates the transaction has been finalized.

§ Conformance

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

EXAMPLE 4

This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

Note, this is an informative note.

§ References

§ Normative References

[RDF-PRIMER]

Frank Manola; Eric Miller. *RDF Primer*. URL: <https://w3c.github.io/rdf-primer/spec/>

[RFC2119]

S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: <https://datatracker.ietf.org/doc/html/rfc2119>

[SHACL]

Holger Knublauch; Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. URL: <https://w3c.github.io/data-shapes/shacl/>

[TRIG]

Gavin Carothers; Andy Seaborne. *RDF 1.1 TriG*. URL: <https://w3c.github.io/rdf-trig/spec/>

[TURTLE]

Eric Prud'hommeaux; Gavin Carothers. *RDF 1.1 Turtle*. URL: <https://w3c.github.io/rdf-turtle/spec/>

[XPATH-FUNCTIONS-31]

Michael Kay. *XPath and XQuery Functions and Operators 3.1*. 21 March 2017. REC. URL: <https://www.w3.org/TR/xpath-functions-31/>

§ Issues Index

ISSUE 1 More extensions should be specified w.r.t. [HTTP status codes](#), or [keeping state](#). This should be further detailed in a chapter after the overview.

ISSUE 2 We should refer here to a new next chapter on how to gracefully iterate over the pages and how to keep the state in more detail cfr. the extensions in Issue 1. We can then also indicate that a client MAY implement the text on [pruning branches](#) related to interpreting comparators for `xsd:dateTime` literals if it wants to detect immutable pages via the `timestampPath`.

ISSUE 3 More specific server documentation should be found in a Server Primer (to do), such as containing a [link to the JSON-LD context](#), [official SHACL shapes for LDES](#) to validate your pages, best practices for publishing an LDES for reaching an optimal performance, best practices for enveloping your data using named graphs, how to build a status log for the use case of an aggregator or harvester, etc.