



30 April  
2025

# SEMIC LDES 2025 A new standardisation trajectory

interoperable  
europe

# SEMIC Assets



Vocabularies



Technical

Application Profiles



# #LDES

# Upcoming LDES Events

## LDES Community Events

22 May

LDES  
Community Workshop

10:00 – 12:00 CEST

25-26  
November

Meet us at

SEMIC<sup>20</sup>  
conference 25

## LDES Standardisation Track

30 April

LDES Working Group  
Onboarding Session

14:00 – 16:00 CEST

7 May

LDES  
Working Group I

14:00 – 16:00 CEST

12 June

LDES  
Working Group II

14:00 – 16:00 CEST

3 July

LDES  
Working Group III

14:00 – 16:00 CEST

# Communication channels

---



- Online meetings (see previous slide)
- Physical meetings: SEMIC2025
- Core discussions documented in Github issues:  
<https://github.com/SEMICeu/LinkedDataEventStreams/issues/>
- Matrix channel:  
<https://matrix.to/#/#ldes:chat.semantic.works>
- Want to follow up all the working groups and receive the reports?  
Leave your e-mail address in the chat!

# Agenda



14:05 - 14:15	Welcome & Tour de table
14:15 - 14:45	The new trajectory & plan for the workshops
14:45 - 15:15	Q&A
15:15 - 16:00	A technical walk-through of the current LDES spec & why we're going to take on certain issues



# Who's attending?

Tour de table

#LDES

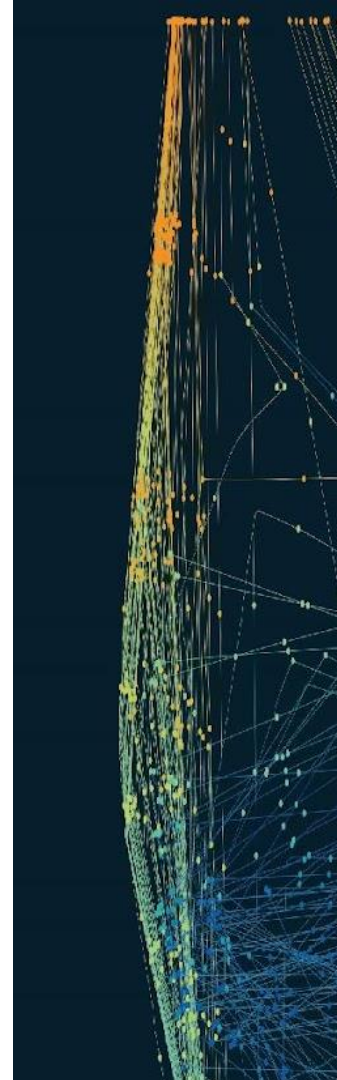






# The 2025 trajectory

#LDES



# The spec today

## A living document

A vocabulary with terms and usage notes for producers and consumers  
So far there has not been a real promise to keep the spec stable

## 30 issues open on the repository

Requests for clarity as well as feature requests.

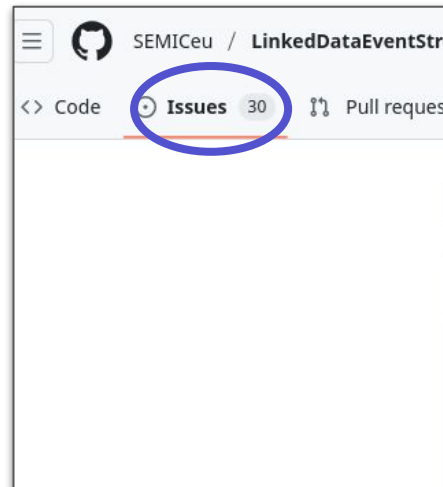
## Implementations in toolchains

- The Flanders Smart Data Space  
<https://informatievlaanderen.github.io/VSDS-Tech-Docs/>
- RDF Connect  
<https://github.com/rdf-connect/ldes-client>
- IncRML: incremental RDF mapping language to LDES  
<https://rml.io/yarrrml/spec/incrml/>
- Semantic Works LDES consumer  
<https://github.com/redpencilio/lides-consumer-service>
- LDES in Solid  
<https://github.com/woutslabbinck/VersionAwareLDESinLDP>

Plenty of custom implementations, such as

[Rijksmuseum](#), [Ocean data](#), a DCAT-AP Feed through [Github Actions](#), ...

# #LDES







## Ambition of this trajectory

A consumer-oriented backwards-compatible **stable release**.

We don't want to write a client for one specific server

Instead, we want to establish a **loose coupling**, and  
write many possible servers for many possible clients.

We want consumers to become unambiguously **testable**.

# Consumer-oriented spec: a new structure



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

The new LDES spec will be structured as follows:

1. An **overview** of the LDES terms and intended use
2. Explaining what a consumer **MUST** implement: initialization, traversing, state management, handling HTTP error codes, ...
3. Interpreting retention policies
4. The vocabulary and a definition of the semantics of each term

**#LDES**

# And an opinionated server primer



At <https://w3id.org/ldes/server-primer> (proposed)

Non-normative, but contains best practices  
applying how a consumer will interpret what has been published

## **Use case focusing on harvesting**

such as data.europa.eu, RINF, or Europeana; also relevant for base registries.

## **Principles**

- High performance by default, not after optimization
- Primer remains domain agnostic
- Explaining how to “envelope” your data

# #LDES

# The server primer is a common basis for the implementation reports



At <https://github.com/SEMICeu/LDES-implementation-reports/>

We'll have implementation reports about:

- DCAT-AP Feeds
- Cultural Heritage Feeds (PR open)
- *Your implementation report? Pull request this today!*

Explains the activity-based model of your domain:

e.g., a `dcat:Dataset` can be created, updated or can be deleted.

## #LDES

# In preparation of this trajectory

A preparation was done in 2024 by Digital Flanders,  
as a result of the experiences when building their toolchain.

[Input document](#)

⇒ this was processed into this presentation, workplan and issue list

# In 2025



## LDES Standardisation Track



A public review period for implementations from 3rd of July until SEMIC2025 where we want to publish a stable LDES spec.

# #LDES



# How will we reach **consensus**?

Everyone prepares comments on the PRs before the meeting.

Decision during the call can be:

1. Accept and merge
2. Conditional accept: merge when conditions are validated
3. Full reject

# Workshops and issues



## **WS1: Restructuring the spec and extending retention policies**

PRs ready for your comments today:

<https://github.com/SEMICeu/LinkedDataEventStreams/pulls>

## **WS2: Consumer algorithm: iteration and state management**

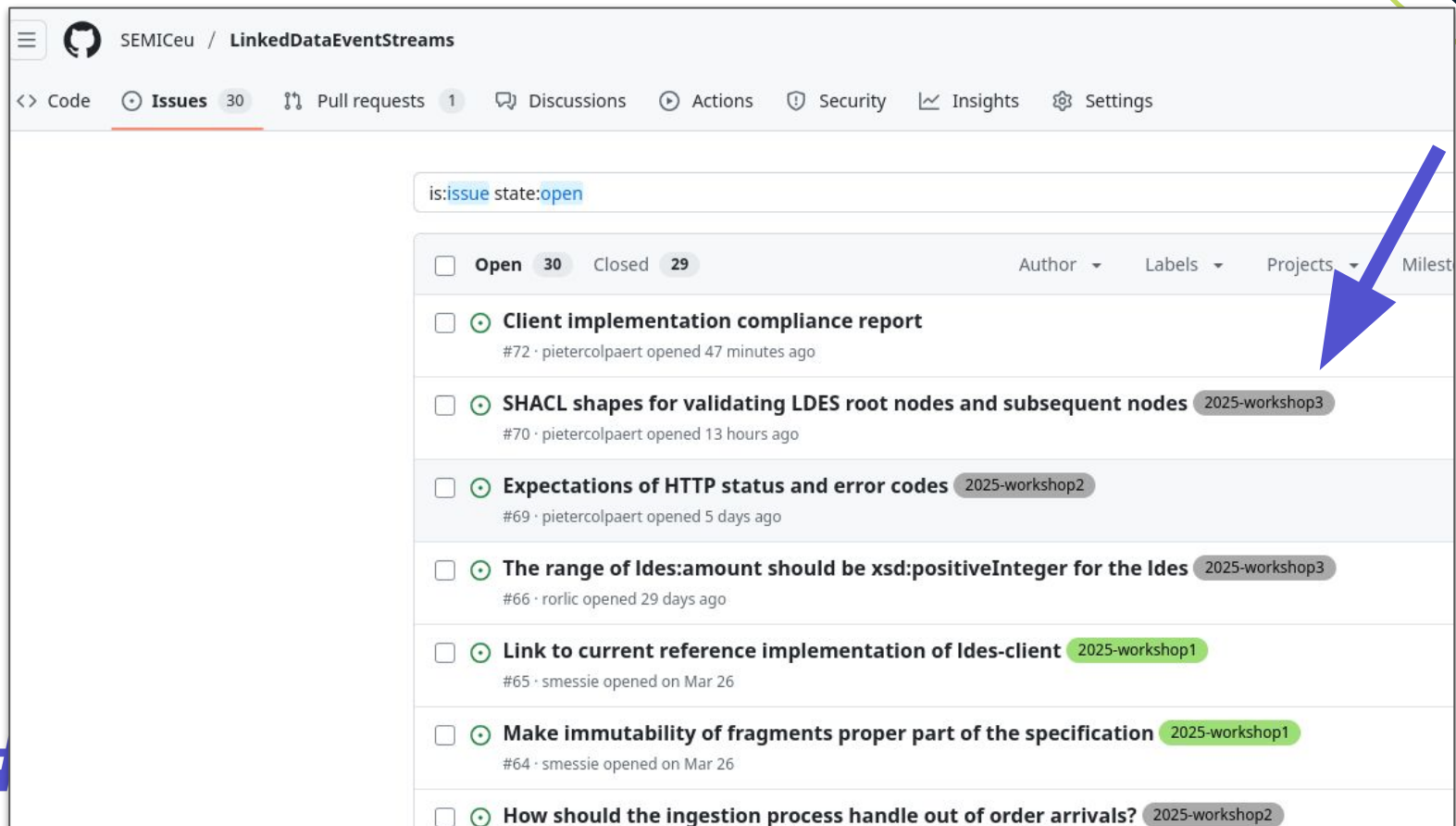
PRs will be ready 1 week before the workshop and we'll send out an email

## **WS3: A server primer with best practices**

PRs will be ready 1 week before the workshop and we'll send out an email

**#LDES**

# Github issues were tagged



SEMICeu / LinkedDataEventStreams

<> Code Issues 30 Pull requests 1 Discussions Actions Security Insights Settings

is:issue state:open

☐ Open 30 Closed 29 Author Labels Projects Milestones

- ☐ **Client implementation compliance report** #72 · pietercolpaert opened 47 minutes ago
- ☐ **SHACL shapes for validating LDES root nodes and subsequent nodes** 2025-workshop3 #70 · pietercolpaert opened 13 hours ago
- ☐ **Expectations of HTTP status and error codes** 2025-workshop2 #69 · pietercolpaert opened 5 days ago
- ☐ **The range of Ides:amount should be xsd:positiveInteger for the Ides** 2025-workshop3 #66 · rorlic opened 29 days ago
- ☐ **Link to current reference implementation of Ides-client** 2025-workshop1 #65 · smessie opened on Mar 26
- ☐ **Make immutability of fragments proper part of the specification** 2025-workshop1 #64 · smessie opened on Mar 26
- ☐ **How should the ingestion process handle out of order arrivals?** 2025-workshop2

# Workshop 1



1. Preparing a rewrite of the spec towards a consumer perspective:
  - a. Overview: defaulting to using named graphs in examples
  - b. The vocabulary of LDES at the end
2. Proposed additions to the overview
  - a. `ldes:EventSource`: the view to select when multiple are available
  - b. Introduction of simple transactions support
  - c. `ldes:immutable` and `ldes:timeToLive` for overruling HTTP caching directives
  - d. clear semantics for `ldes:timestampPath`
  - e. add `tree:member` MUST be an IRI and not a blank node
  - f. sequence numbers?
3. Retention policies
  - a. better description of existing retention policies
  - b. transaction awareness in retention policies
  - c. having a separate policy on types: a deletion
  - d. adding text on combining multiple retention policies
4. Vocabulary changes
  - a. Fixing datatypes (e.g., `xsd:nonNegativeInteger` for `ldes:amount` should just be `xsd:integer` – the constraint should then however be a SHACL constraint: `ldes:amount` needs to be `>0`)
  - b. New terms for immutability of nodes and for transactions – see overview
  - c. A formal definition of `ldes:EventSource`



LDES  
Working Group I

14:00 – 16:00 CEST

# Workshop 1 fixes these open GH issues

- Definition of `ldes:timestampPath` and the fact out-of-order is impossible: [#10](#), [#35](#), [#49](#), [#61](#)
- Definition of `ldes:EventSource`: [#34](#)
- named graphs and activity streams-like examples in the spec: [#37](#) , [#43](#)
- `tree:member` in LDES points to an IRI, not a blank node [#56](#)
- Retention policies:
  - LatestVersionSubset: better description [#47](#)
  - Retention policies with a deletion [#50](#)
- `ldes:immutable` and `ldes:timeToLive` on a `tree:Node`: [#53](#)



LDES  
Working Group I

14:00 – 16:00 CEST

# #LDES

# A PR was opened as a basis for solving these issues

<https://github.com/SEMICeu/LinkedDataEventStreams/pull/71>

# #LDES

## First full review towards a client oriented specification #71

Open pietercolpaert wants to merge 1 commit into master from workshop1

Conversation 0 Commits 1 Checks 0 Files changed 4

pietercolpaert commented 38 minutes ago • edited Member

This PR is the basis for the workshops: it rewrites the spec towards a client-oriented perspective. When features will be added, they should be added on top of this rewrite.

Changes done:

- Derived collections are gone from the spec: the terms remain in the vocabulary turtle file for now, but will not any more be advertised. If there's any need in the future, we can reboot them in a future server primer (workshop 3).
- There's now an LDES vocabulary section at the bottom with semantics, subclassing, etc.
- An introduction and overview section was written from scratch, including examples with named graphs by default. Versioned identifiers are now less important. There is also a
- The concept of `ViewDescriptions` or `View` are moved back out the spec: it was something that did not end up properly in the TREE specification after all as it caused confusion. The only idea that is left is the `tree:viewDescription` that will remain useful.
- `ldes:EventSource` has been added as a concept to the Vocabulary
- `tree:importStream` is now gone; this wasn't used in any algorithm at this point and probably needs an LDES specific way of handling this. Also imports have been removed from the stable TREE spec, and this was considered experimental.
- Retention policies were rewritten towards a client perspective
- All `xsd:dateTime` literals in retention policies must have a timezone
- An image was added with an overview of the retention policies

This PR fixes these issues:

- Definition of `ldes:timestampPath` and the fact out-of-order is impossible: [Motivate streams not ordered by publish time #10](#) [Clarify MAY on ldes:timestampPath #35](#) [Add note on version objects #49](#) [What are the semantics of the ldes:timestampPath ? #61](#)
- Definition of `ldes:EventSource` [Proposal: further constraining the EventSource view #24](#)





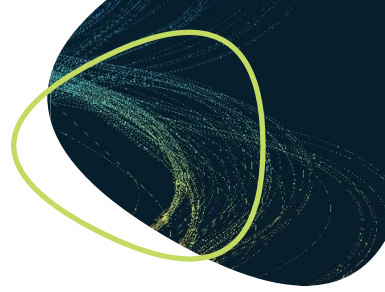
# We need your help

Please review the PRs by next week

During the workshop, we'll walk through the issues, the PRs and their comments

**#LDES**





What's planned for WS2 and WS3?

# Workshop 2: Iterating and state management

1. “Fragmenting and pagination” chapter 2 in the spec needs to become “iterating and state management”
2. A high-level algorithm and a standardized state management
3. Expected behaviour of a consumer for Error handling of HTTP codes
4. Retention policies: what must happen when retention policies conflict with the consumer configuration? An error needs to be given when the polling happens too slowly.



LDES  
Working Group II

14:00 – 16:00 CEST

#LDES

# Workshop 2 will fix these Github issues

- Standardized client iterator: [#31](#)
- Supporting a streaming profile of LDES: [#42](#)
- Expected behaviour on HTTP status codes: [#69](#)



LDES  
Working Group II

14:00 – 16:00 CEST

**#LDES**

# Workshop 3: the server primer

- Best practices in the server primer says that this `ldes:EventSource` is best structured using a chronological search tree on `timestampPath`
- Creating 2 SHACL shapes for an LDES: for the root node and for any subsequent node
- Best practices for the use case of replication/synchronization on how to build a high-performance LDES server, elaborating on various features with a.o. the TREE profile: indicating a streaming profile
- Consumer status log
- Note on relative IRIs: desired
- Log compaction with retention policies
- Provenance (We should move the text on version materialization here)
- Searching for LDESs through a DCAT catalog: best practices
- Best practices for a signing the members in an LDES and/or providing sticky policies for them
- How to manage a derived LDES



LDES  
Working Group III

14:00 – 16:00 CEST

#LDES

# Workshop 3: Fixing these open issues

- Indicating you have a derived LDES: [#44](#)
- Status indication of a derived view: [#5](#)
- Recommendation on using relative IRIs: [#41](#)
- Status log of a consumer service: [#54](#)
- Note on consistent graph replication and how that could be done [#51](#)
- Server publishing data handling out of order arrivals [#63](#)
- Creating a SHACL for the root node and further nodes, fixing things like constraints on literals: [#66](#) [#70](#)



LDES  
Working Group III

14:00 – 16:00 CEST

**#LDES**





# Q&A

Any questions?

#LDES





# Onboarding

A walk through the LDES spec today  
and where we're heading

**#LDES**





# Integrating a data dump is a one-off. Integrating a stream is for a lifetime.

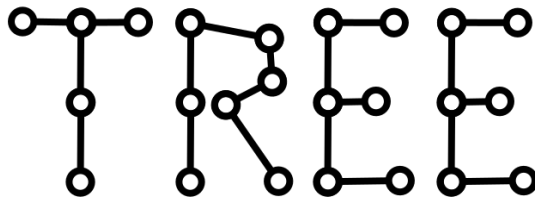
Given a start URL, an LDES client returns a stream of members of the corresponding `ldes:EventStream`.

First, the history that is available from this entry point is emitted, and once the client has caught up with the stream, it remains synchronized as new members are published.



# LDES is built upon TREE hypermedia

A specification by the W3C TREE community group  
with an overlapping community with the SEMIC LDES working group



**#LDES**



[Home](#) / TREE hypermedia Community Grou...

## TREE HYPERMEDIA COMMUNITY GROUP

The TREE hypermedia community group will discuss materializable hypermedia interfaces. Its goals are to: 1. Further evolve the TREE hypermedia specification (<https://w3id.org/tree/specification>) and its vocabulary (<https://w3id.org/tree/>) 2. Create a test suite for spec compliance of both servers and clients 3. Deliver a specification on view definitions for source selection



Group's public email, repo and wiki activity over time



Note: Community Groups are proposed and run by the community. Although W3C hosts these conversations, the groups do not necessarily represent the views of the W3C Membership or staff.

### Drafts / [licensing info](#)

Date	Name
2023-05-25	<a href="#">The TREE hypermedia specification</a>

Chairs, when logged in, may publish draft and final reports. Please see [report requirements](#).

### Call for Participation in TREE hypermedia Community Group

W3C Team | Posted on: March 28, 2023

### Tools for this group

Mailing List

IRC

GitHub

RSS

Contact This Group

### Get involved

Anyone may join this Community Group. All participants in this group have signed the [W3C Community Contributor License Agreement](#).

[JOIN OR LEAVE THIS GROUP](#)



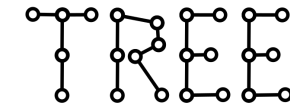
Pieter Colpaert

*Chairs*

### Participants (21)



# #LDES



# To discuss the TREE hypermedia specification



## The TREE hypermedia specification

Draft Community Group Report, 12 April 2025



### ▼ More details about this document

#### This version:

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

#### Feedback:

[public-treecg@w3.org](mailto:public-treecg@w3.org) with subject line "[TREE] ... message topic ..." (archives)

#### Issue Tracking:

[GitHub](#)

#### Editor:

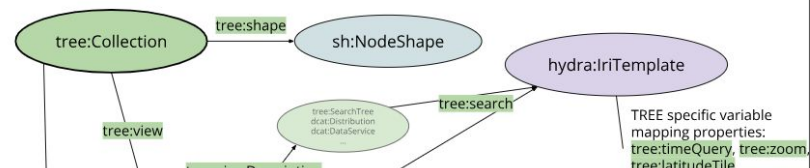
[Pieter Colpaert](#)

Copyright © 2025 World Wide Web Consortium. W3C® liability, trademark and permissive document license rules apply.

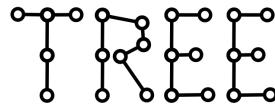
### Abstract

The TREE specification provides instructions for clients to interpret and navigate Web APIs structured as search trees. It defines how members (sets of quads) in a dataset can be distributed across multiple pages interlinked through relationships. The specification introduces key concepts such as `tree:Collection` (a set of members), `tree:Node` (the pages in the search tree), and `tree:Relation` (links between nodes). By interpreting such qualified relations and search forms, TREE enables clients to efficiently retrieve their members of interest.

### § 1. Overview



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



European  
Commission



# An example: a collection with members

```
ex:Collection1 a tree:Collection;  
    tree:view <> ;  
    tree:member ex:Subject1, ex:Subject2 .
```

```
ex:Subject1 a ex:Subject ;  
    rdfs:label "Subject 1" ;  
    ex:value 1 .
```

```
ex:Subject2 a ex:Subject ;  
    rdfs:label "Subject 2" ;  
    ex:value 2 .
```



## 1. Initialisation

```
ex:Collection1 a tree:Collection;  
    tree:view <> ;  
    tree:member ex:Subject1, ex:Subject2 .
```

A client will be looking for the tree:view triple to find the collection this page contains a fragment of.

```
ex:Subject1 a ex:Subject ;  
    rdfs:label "Subject 1" ;  
    ex:value 1 .
```

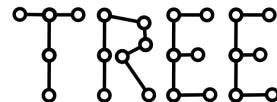
```
ex:Subject2 a ex:Subject ;  
    rdfs:label "Subject 2" ;  
    ex:value 2 .
```

# Features of the TREE spec

Note: we use Turtle / TRiG,  
but works equally as well in JSON-LD

```
{
  "@context": { ... },
  "@id": "ex:Collection1",
  "@type": "tree:Collection",
  "tree:view": {
    "@id": ""
  },
  "tree:member": [
    {
      "@id": "ex:Subject1",
      "@type": "ex:Subject",
      "rdfs:label": "Subject 1",
      "ex:value": 1
    },
    {
      "id": "ex:Subject2",
      "@type": "ex:Subject",
      "rdfs:label": "Subject 2",
      "ex:value": 2
    }
  ]
}
```

#LDES



## 2. Member extraction

```
ex:Collection1 a tree:Collection;  
    tree:view <> ;  
    tree:member ex:Subject1, ex:Subject2 .
```

```
ex:Subject1 a ex:Subject ;  
    rdfs:label "Subject 1" ;  
    ex:value 1 .
```

→ A client may extract all statements related to a member

```
ex:Subject2 a ex:Subject ;  
    rdfs:label "Subject 2" ;  
    ex:value 2 .
```



## 2. Member extraction

```
ex:Collection1 a tree:Collection;  
  tree:view <> ;  
  tree:member ex:Subject1v1, ex:Subject1v2.
```

```
ex:Subject1v1 a dct:isVersionOf ex:Subject1 ;  
  dct:created "2025-04-30T12:00:00Z" .
```

```
ex:Subject1v1 {  
  ex:Subject1 ex:value 1 .  
}
```

```
ex:Subject1v2 {  
  ex:Subject1 ex:value 2 .  
}
```

Which becomes more tedious in more complex examples.

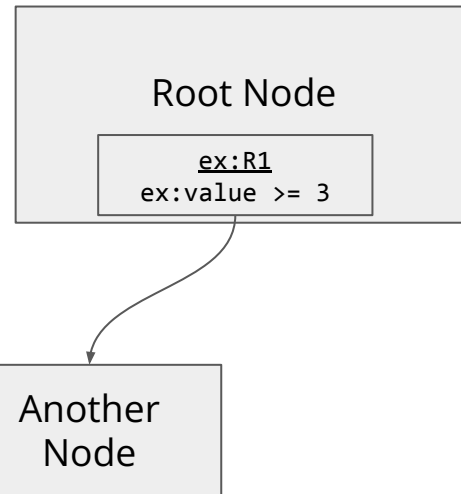
E.g., with named graphs, but also out-of-band members are supported.

There's also possibility for using shape topologies using a SHACL shape.

## 3. Traversing a search tree

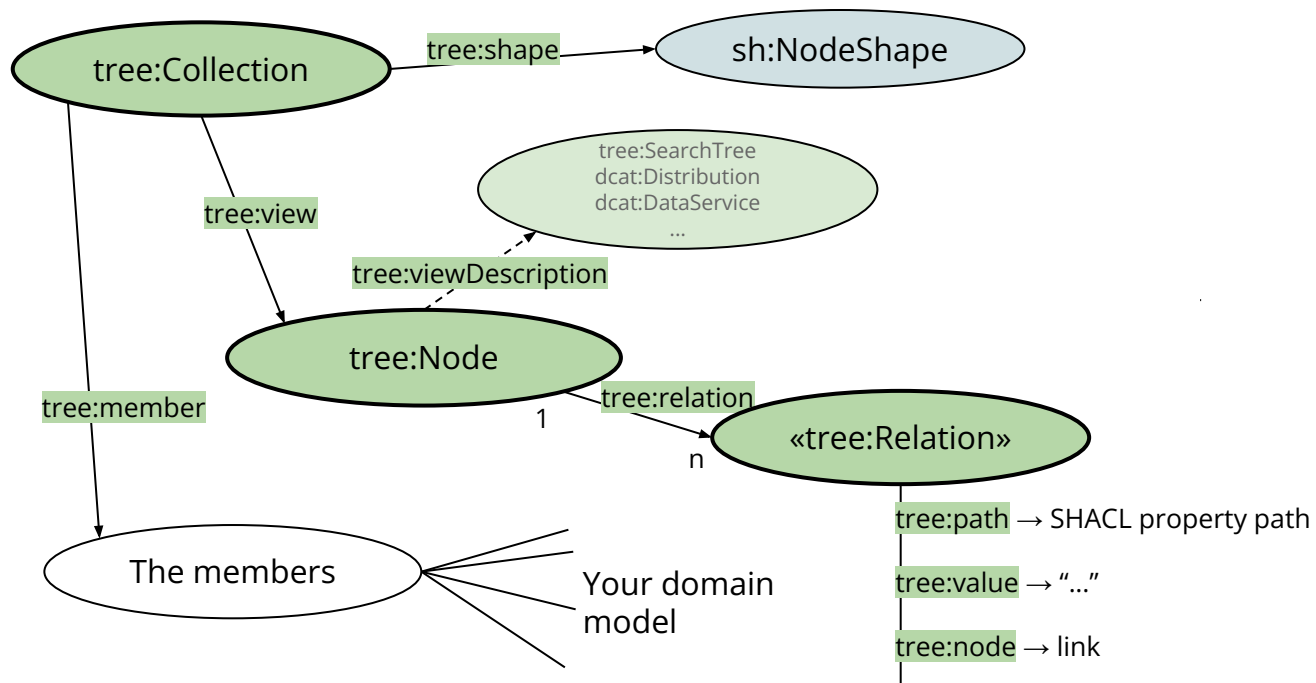
```
ex:Collection1 a tree:Collection;  
    tree:view <> ;  
    tree:member ex:Subject1v1, ex:Subject1v2.
```

```
<> tree:relation ex:R1 .  
  
ex:R1 a tree:GreaterThanOrEqualToRelation ;  
    tree:node ex:AnotherNode ;  
    tree:value 3;  
    tree:path ex:value .
```

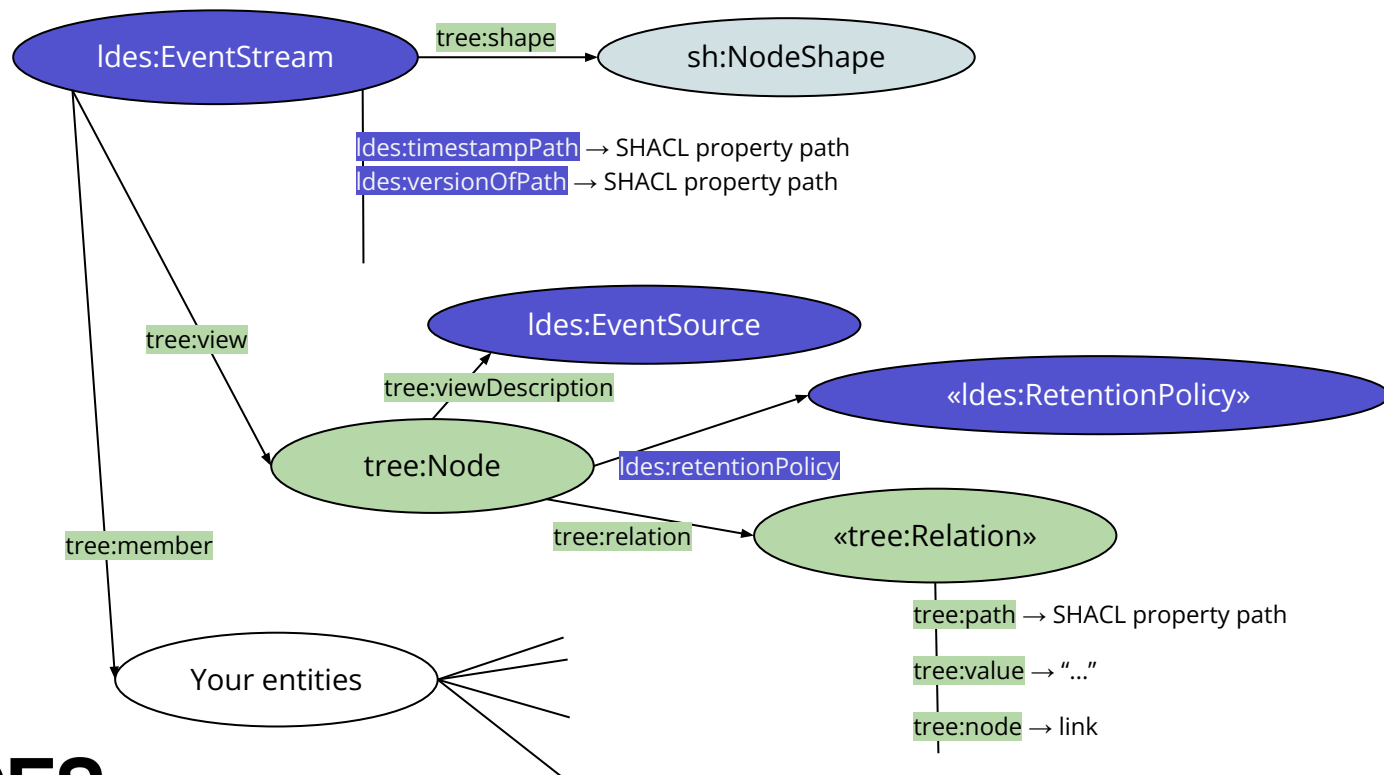


Based on the description, a client can understand whether it wants to follow the relation or not.

# TREE: the overview



# LDES extends TREE with a couple of terms





# An example of an LDES



```
ex:Collection1 a ldes:EventStream;  
    tree:view <> ;  
    tree:shape <shape.ttl> ;  
    ldes:timestampPath dct:created;  
    ldes:versionOfPath dct:isVersionOf;  
    tree:member ex:Subject1v1, ex:Subject1v2 .
```

```
ex:Subject1v1 a dct:isVersionOf ex:Subject1 ;  
    dct:created "2025-04-30T12:00:00Z" .
```

```
ex:Subject1v1 {  
    ex:Subject1 ex:value 1 .  
}
```

```
ex:Subject1v2 a dct:isVersionOf ex:Subject1 ;  
    dct:created "2025-04-30T13:00:00Z" .
```

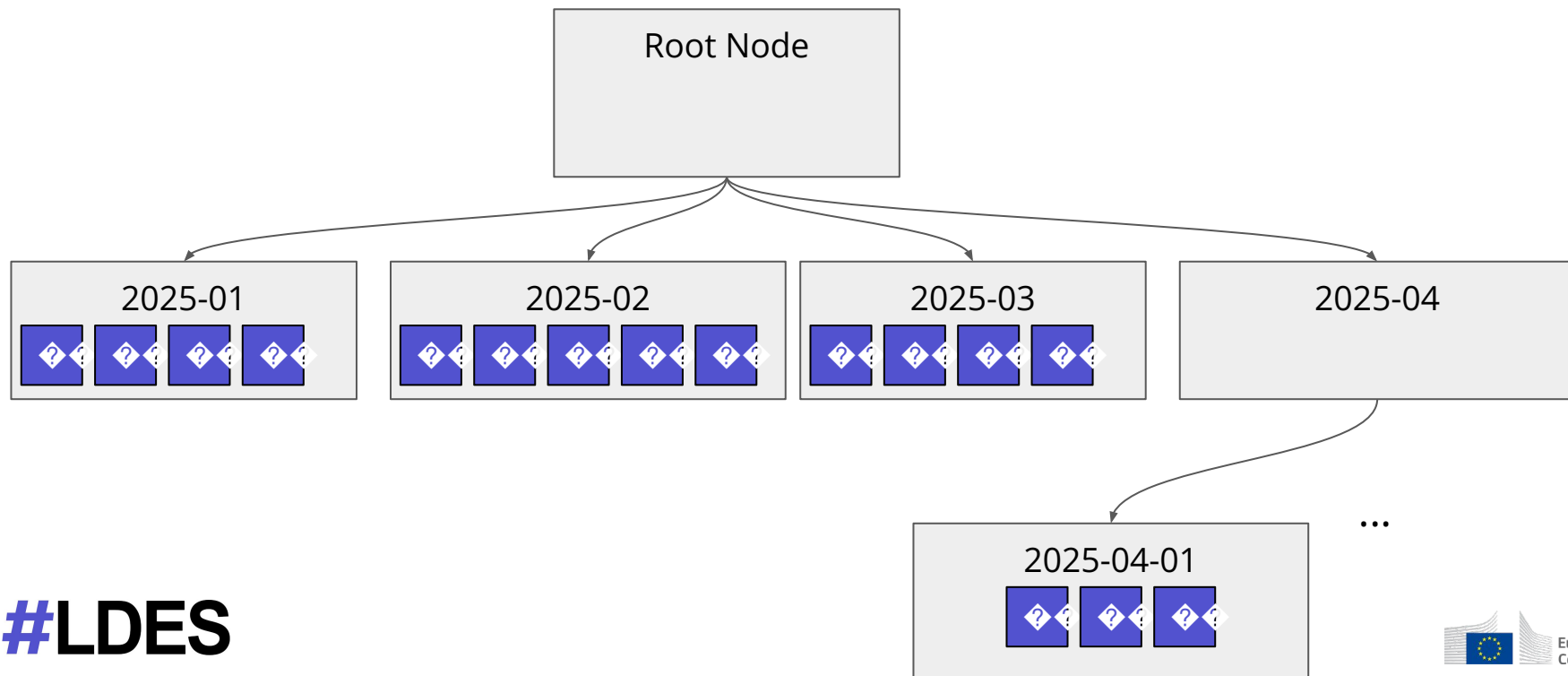
```
ex:Subject1v2 {  
    ex:Subject1 ex:value 2 .  
}
```



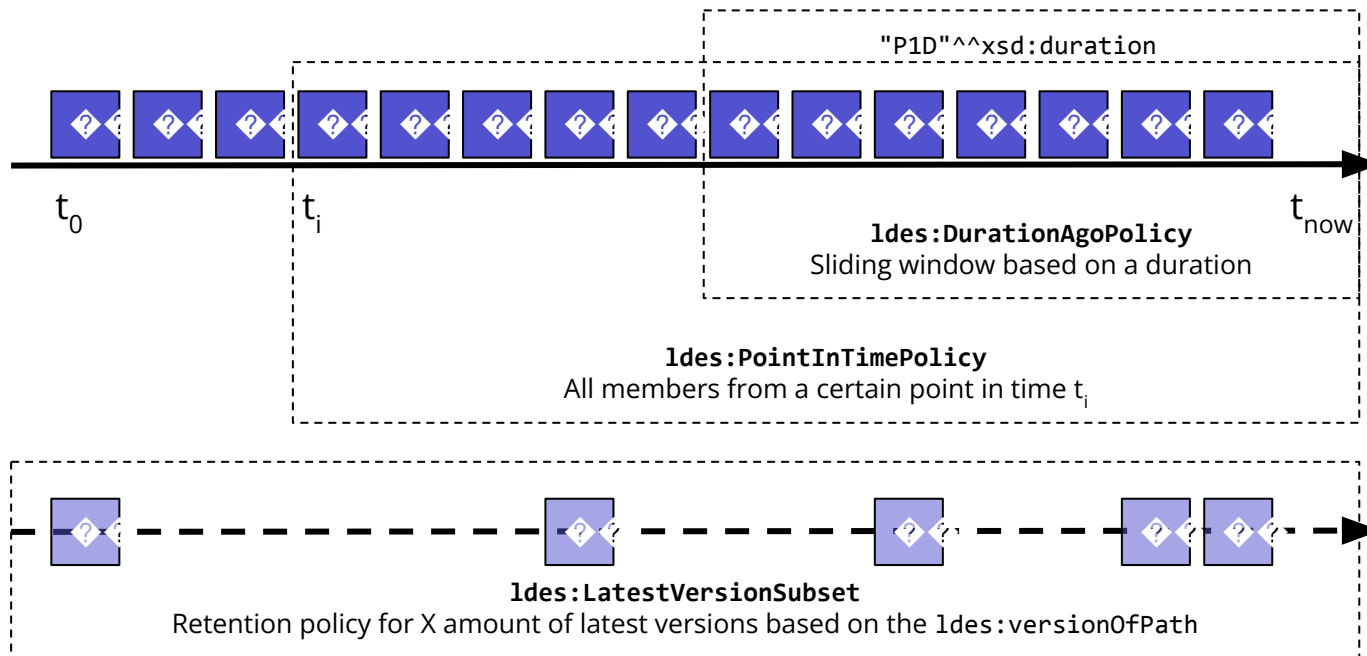
# What if this page grows too large?

1. Fragmentations
2. Retention policies

# Using TREE relations to fragment an event stream as a chronological search tree



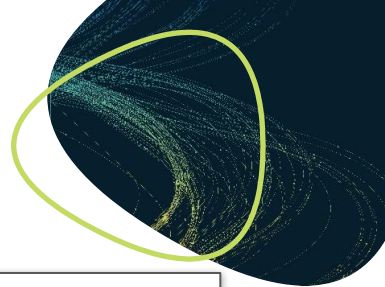
# LDES retention policies





# A walk through the LDES spec today

**#LDES**



## Abstract

A Linked Data Event Stream is a collection of immutable objects (such as version objects, sensor observations or archived representations). Each object is described in RDF.

**Now:** The abstract currently defines what an LDES is.

**Should become:** specifying why you should read the spec any further – the fact that this spec will define how you can build a client.

**#LDES**

## § 1. Introduction

A Linked Data Event Stream (LDES) (`ldes:EventStream`) is a collection (`rdfs:subClassOf tree:Collection`) of immutable objects, each object being described using a set of RDF triples ([\[rdf-primer\]](#)).

This specification uses the [TREE specification](#) for its collection and fragmentation (or pagination) features, which in its turn is compatible to other specifications such as [\[activitystreams-core\]](#), [\[VOCAB-DCAT-2\]](#), [\[LDP\]](#) or [Shape Trees](#). For the specific compatibility rules, read the [TREE specification](#).

Note: When a client once processed a member, it should never have to process it again. A Linked Data Event Stream client can thus keep a list of (or cache) already processed member IRIs. A reference implementation of a client is available as part of the Comunica framework on [NPM](#) and [Github](#).

- Defines the term EventStream (overview/vocabulary)
- Says it relies on the TREE specification and you should check that out, and the fact that this would be compatible with other specifications
- Note: This reference implementation is deprecated by now, and a new one is available.

# The TREE specification

LDES is built on top of W3C TREE CG specifications.

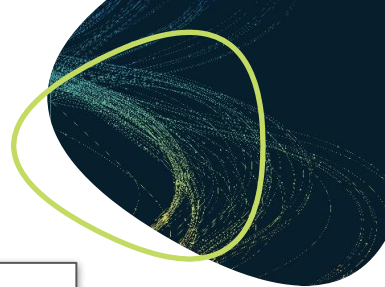
So we need to build a TREE client then first... The TREE spec also went through the same evolution: from a producer primer to a consumer specification.

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

Upcoming: a discovery specification at

<https://w3id.org/tree/specification/discovery>





The base URI for LDES is <https://w3id.org/ldes#>, and the preferred prefix is `ldes:`. Other prefixes are used following [prefix.cc](#).

#### EXAMPLE 1

```
ex:C1 a ldes:EventStream ;
      ldes:timestampPath sosa:resultTime ;
      tree:shape ex:shape1.shacl ;
      tree:member ex:Observation1 .

ex:Observation1 a sosa:Observation ;
                sosa:resultTime "2021-01-01T00:00:00Z"^^xsd:dateTime ;
                sosa:hasSimpleResult "..."
```



The `ldes:EventStream` instance SHOULD have these properties:

- `tree:shape`: the shape of the collection defines its members. It tells clients all old and new members of the stream have been and will be validated by that shape. As a consequence of the immutability of the members, this shape MAY evolve, but it MUST always be backwards compatible to the earlier version.
- `tree:member` indicating the members of the collection.

The `ldes:EventStream` instance MAY have these properties:

- `ldes:timestampPath` indicating how you can understand using a timestamp (`xsd:dateTime`) a member precedes another member in the LDES
- `ldes:versionOfPath` indicating the non-version object (see example below).



## EXAMPLE 2

```
ex:C2 a ldes:EventStream ;
    ldes:timestampPath dcterms:created ;
    ldes:versionOfPath dcterms:isVersionOf ;
    tree:shape ex:shape2.shacl ;
    tree:member ex:AddressRecord1-version1 .

ex:AddressRecord1-version1 dcterms:created "2021-01-01T00:00:00Z"^^xsd:dateTime ;
    adms:versionNotes "First version of this address" ;
    dcterms:isVersionOf ex:AddressRecord1 ;
    dcterms:title "Streetname X, ZIP Municipality, Country" .
```

Note: When you need to change an earlier version of an `ldes:EventStream`, there are two options: create a new version of the object with a new shape that is backward compatible, and add the new version of that object again as a member on the stream, or replicate and transform the entire collection into a new `ldes:EventStream`. You can indicate that the new `ldes:EventStream` is derived from another `ldes:EventStream`.

Note: in Example 1, we consider the Observation object to be an immutable object and we can use the existing identifiers. In Example 2 however, we still had to create version IRIs in order to be able to link to immutable objects.

# Versioned identifiers are a headache to handle

Thanks to TREE, we now however also support named graphs. E.g.,

```
<C1> a ldes:EventStream ;  
      tree:member <streetname1-v1>, <streetname1-v2> .  
<streetname1-event1> a as:Create ; # using the Activity Streams vocabulary  
                        as:object <streetname1> ;  
                        as:published "2026-01-01T00:10:00Z"^^xsd:dateTime .  
<streetname1-event1> {  
  <streetname1> rdfs:label "Station Road" ;  
                ex:locatedIn <municipalityname> .  
  <municipalityname> rdfs:label "Ghent" ;  
}
```

# Elaborates on how to apply the TREE spec



## § 2. Fragmenting and pagination

The focus of an LDES is to allow clients to replicate the history of a dataset and efficiently synchronize with its latest changes. Linked Data Event Streams MAY be fragmented when their size becomes too big for 1 HTTP response. Fragmentations MUST be described using the features in the [TREE specification](#). All relation types from the TREE specification MAY be used.

### EXAMPLE 3

```
ex:C1 a ldes:EventStream ;
    ldes:timestampPath sosa:resultTime ;
    tree:shape ex:shape1.shacl ;
    tree:member ex:Observation1, ... ;
    tree:view <?page=1> .

<?page=1> a tree:Node ;
    tree:relation [
        a tree:GreaterThanEqualToRelation ;
        tree:path sosa:resultTime ;
        tree:node <?page=2> ;
        tree:value "2020-12-24T12:00:00Z"^^xsd:dateTime
    ] .
```

# More server primer like notes follow



An `tree:importStream` MAY be used to describe a publish-subscribe interface to subscribe to new members in the LDES.

Note: A 1-dimensional fragmentation based on creation time of the immutable objects is probably going to be the most interesting and highest priority fragmentation for an LDES, as only the latest page, once replicated, should be subscribed to for updates. However, it may happen that a time-based fragmentation cannot be applied. For example: the backend system on which the LDES has been built does not receive the events at the time they were created, due to human errors (forgetting to indicate that a change was made), external systems or just latency. Applying a time-based fragmentation in that situation will result in losing caching, due to the ever-changing pages. Instead, in the spirit of an LDES's goal, the publisher should publish the events in the order they were received by the backend system (that order is never changing), trying to give as many pages as possible an `HTTP Cache-Control: public, max-age=604800, immutable` header.

Note: Cfr. [the example in the TREE specification on "searching through a list of objects ordered in time"](#), also a search form can optionally make a one dimensional feed of immutable objects more searchable.

Multiple problems with this text today however:

1. `tree:importStream` has never been implemented and was removed from the TREE spec
2. Note 1:
  - a. We now know a 1-dimensional pagination is not at all the best way to publish an LDES.  
A Chronological search tree is also very easy to implement and comes with a lot of benefits
  - b. We have learned a lot of other good practices for servers in the meantime (Working Group 3)
3. Note 2: Using `hydra:search` is a patch for the 1-dimensional pagination that is not needed with a chronological search tree.

## #LDES



### § 3. Retention policies

By default, an LDES MUST keep all members that has been added to the `ldes:EventStream`. It MAY add a retention policy in which the server indicates data will be removed from the server. Third parties SHOULD read retention policies to understand what subset of the data is available in this `tree:View`, and MAY archive these members.

In the LDES specification, three types of retention policies are defined which can be used with a `ldes:retentionPolicy` with an instance of a `tree:View` as its subject:

1. `ldes:DurationAgoPolicy`: a time-based retention policy in which data generated before a specified duration is removed
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 removed

Different retention policies MAY be combined. When policies are used together, a server MUST store the members as long they are not all matched.

- `tree:View` was never introduced in TREE in the end and the range of `tree:viewDescription` was left open. Recommended is to add retentionPolicies mainly on the root node of a search tree now.
- We will also need support for intersecting retention policies, and not only taking the union of it

# What is the functionality a client must implement based on a retention policy?



1. Checking whether you are fast enough to get a valid replication  
⇒ needs to be part of the client algorithm
2. Source selection ⇒ but this algorithm is not specified or not part of the scope of this standardization trajectory. See TREE Discovery nonetheless.

Today however, the main purpose is metadata to manually select the right search tree you want to use for a particular use case.



### § 3.1. Time-based retention policies

A time-based retention policy can be introduced as follows:

#### EXAMPLE 4

```
ex:C3 a ldes:EventStream ;
      ldes:timestampPath prov:generatedAtTime ;
      tree:view <> .

<> ldes:retentionPolicy ex:P1 .

ex:P1 a ldes:DurationAgoPolicy ;
      tree:value "P1Y"^^xsd:duration . # Keep 1 year of data
```

A `ldes:DurationAgoPolicy` uses a `tree:value` with an `xsd:duration`-typed literal to indicate how long ago the timestamp, indicated by the `ldes:timestampPath` that MAY be redefined in the policy itself.

## § 3.2. Version-based retention policies

### EXAMPLE 5

In order to indicate you only keep 2 versions of an object referred to using `dcterms:isVersionOf`:

```
ex:C2 a ldes:EventStream ;
      ldes:timestampPath dcterms:created ;
      ldes:versionOfPath dcterms:isVersionOf ;
      tree:view <> .

<> ldes:retentionPolicy ex:P2 .

ex:P2 a ldes:LatestVersionSubset;
      ldes:amount 2 ;
      #If different from the Event Stream, this can optionally be overwritten here
      ldes:timestampPath dcterms:created ;
      ldes:versionOfPath dcterms:isVersionOf .
```

A `ldes:LatestVersionSubset` MUST define the predicate `ldes:amount` and MAY redefine the `ldes:timestampPath` and/or `ldes:versionOfPath`. It MAY also define a compound version key using `ldes:versionKey` (see example below) instead of the more `ldes:versionOfPath`. The `ldes:amount` has a `xsd:nonNegativeInteger` datatype and indicated how many to keep that defaults to 1. The `ldes:versionKey` is an `rdf:List` of SHACL property paths indicating objects that MUST be concatenated together to find the key on which versions are matched. When the `ldes:versionKey` is set to an empty path `()`, all members MUST be seen as a version of the same thing.

### EXAMPLE 6

For sensor datasets the version key may get more complex, grouping observations by both the observed property as the sensor that made the observation.

```
ex:C1 a ldes:EventStream ;
    tree:view <> .

<> ldes:retentionPolicy ex:P3 .

ex:P3 a ldes:LatestVersionSubset;
    ldes:amount 2 ;
    ldes:versionKey ( ( sosa:observedProperty ) ( sosa:madeBySensor ) ) .
```

### § 3.3. Point-in-time retention policies

A point-in-time retention policy can be introduced as follows:

#### EXAMPLE 7

```
ex:C4 a ldes:EventStream ;
      ldes:timestampPath prov:generatedAtTime ;
      tree:view <> .

<> ldes:retentionPolicy ex:P4 .

ex:P4 a ldes:PointInTimePolicy ;
      ldes:pointInTime "2023-04-12T00:00:00"^^xsd:dateTime . # Keep data after April 12t
```

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, indicated by the `ldes:timestampPath` that MAY be redefined in the policy itself.

# More complex retention policies needed

- Based on transactions
- Different retention policy for deletions
- Composite retention policies taking the intersection of multiple



Was the idea to describe a dataset an interpretation of an event stream

## § 4. Derived collections

We will extend the spec with multiple best practices on how to annotate that your newly published collection is derived from an LDES.

First we talk about a versioned LDES. Versioned LDESeS allow for changing an object in an `ldes:EventStream`, while maintaining the history of events. It is achieved by defining change in an `ldes:EventStream` through new `tree:member` in the `ldes:EventStream` through added metadata for both the `ldes:EventStream` and each `tree:member`.

Secondly, version materializations are defined that use a versioned LDES as a basis. This technique allows to create **snapshots** in time of a versioned LDES. Here we define a **snapshot** as `tree:Collection` of the most recent versions of all objects in the versioned LDES.



# A PR was opened as a basis for solving these issues

Please help reviewing this by next week

<https://github.com/SEMICeu/LinkedDataEventStreams/pull/71>

# #LDES

## First full review towards a client oriented specification #71

 Open **pietercolpaert** wants to merge 1 commit into `master` from `workshop1` 

 Conversation 0  Commits 1  Checks 0  Files changed 4



**pietercolpaert** commented 38 minutes ago • edited

Member ...

This PR is the basis for the workshops: it rewrites the spec towards a client-oriented perspective. When features will be added, they should be added on top of this rewrite.

Changes done:

- Derived collections are gone from the spec: the terms remain in the vocabulary turtle file for now, but will not any more be advertised. If there's any need in the future, we can reboot them in a future server primer (workshop 3).
- There's now an LDES vocabulary section at the bottom with semantics, subclassing, etc.
- An introduction and overview section was written from scratch, including examples with named graphs by default. Versioned identifiers are now less important. There is also a
- The concept of `ViewDescriptions` or `View` are moved back out the spec: it was something that did not end up properly in the TREE specification after all as it caused confusion. The only idea that is left is the `tree:viewDescription` that will remain useful.
- `ldes:EventSource` has been added as a concept to the Vocabulary
- `tree:importStream` is now gone; this wasn't used in any algorithm at this point and probably needs an LDES specific way of handling this. Also imports have been removed from the stable TREE spec, and this was considered experimental.
- Retention policies were rewritten towards a client perspective
- All `xsd:dateTime` literals in retention policies must have a timezone
- An image was added with an overview of the retention policies

This PR fixes these issues:

- Definition of `ldes:timestampPath` and the fact out-of-order is impossible: [🔗 Motivate streams not ordered by publish time #10](#) [🔗 Clarify MAY on ldes:timestampPath #35](#) [🔗 Add note on version objects #49](#) [🔗 What are the semantics of the ldes:timestampPath ? #61](#)
- Definition of `ldes:EventSource` [🔗 Proposal: further constraining the EventSource view #24](#)

- 1 Introduction
- 2 Overview
- 3 Retention policies
  - 3.1 Time-based retention policies
  - 3.2 Version-based retention policies
  - 3.3 Point-in-time retention policies
- 4 Vocabulary
  - 4.1 Ides:EventStream
  - 4.2 Ides:timestampPath
  - 4.3 Ides:versionOfPath
  - 4.4 Ides:EventSource
  - 4.5 Ides:RetentionPolicy
  - 4.6 Ides:retentionPolicy
  - 4.7 Ides:DurationAgoPolicy
  - 4.8 Ides:LatestVersionSubset
    - 4.8.1 Ides:versionKey
    - 4.8.2 Ides:amount
  - 4.9 Ides:PointInTimePolicy
    - 4.9.1 Ides:pointInTime

#### Conformance

#### References

Normative References

#### Issues Index

# Linked Data Event Streams

Living Standard, 29 April 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 29 April 2025, the editors have made this specification available under the [Open Web Foundation Agreement Version 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 in RDF using a set of statements. 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.

## § 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](#). The client discovers and follows `tree:Relation` to traverse all pages in the search tree publishing the event stream. Thanks to caching directives and timestamp indications, it will understand which pages it will need to fetch again at a later time to stay synchronized.



# Communication channels



- Online meetings: next one is next week Wednesday same time
- Physical meetings: SEMIC2025
- Core discussions documented in Github issues:  
<https://github.com/SEMICeu/LinkedDataEventStreams/issues/>
- Matrix channel:  
<https://matrix.to/#/#ldes:chat.semantic.works>
- Want to follow up all the working groups and receive the reports?  
Leave your e-mail address in the chat!

Thanks for joining and see you next week for getting  
some standardisation work done!

**#LDES**



# Every time a term gets added

- must be added in the JSON-LD context
- must be added in the vocabulary ttl
- must be added in the vocabulary.md
- may be added in the overview
- must be added in the SHACL shapes

