ETSI GR CIM 056 V1.1.1 (2025-02)

Context Information Management (CIM); Handling of Linked Data Event Streams with NGSI-LD

Disclaimer

The present document has been produced and approved by the cross-cutting Context Information Management (CIM) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGR/CIM-0056

Keywords
API, architecture, data interoperability, data sharing, NGSI-LD

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) cross-cutting Context Information Management (CIM).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document investigates how LDES sources can be consumed by NGSI-LD context brokers and vice versa how NGSI-LD context brokers can make their context information available to LDES consumers.

1 Scope

The present document brings an introduction of the Linked Data Event Streams (LDES) and TREE specifications and maps part of its key concepts to the NGSI-LD model. Also, a mapping of the NGSI-LD entity representation to LDES members is proposed.

The second part of the present document focuses on the architectural considerations of consuming and publishing LDES with NGSI-LD context brokers.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] "Linked Data Event Streams Specification".

[i.2] W3C®: "The TREE hypermedia specification".

[i.3] ETSI GS CIM 009 (V1.8.1): "Context Information Management (CIM); NGSI-LD API".

[i.4] OGC®: "Web Feature Service".

[i.5] OGC® APIs.

[i.6] W3C® Recommendation 21 March 2013: "SPARQL 1.1 Query Language".

[i.7] Linked Data Event Streams Server (Flemish Smart Data Space).

[i.8] Linked Data Event Streams JSON-LD context.

[i.9] Slippy map tilenames.

[i.10] Pagination (Flemish Smart Data Space).

[i.11] Reference Fragmentation (Flemish Smart Data Space).

[i.12] Geospatial fragmentation (Flemish Smart Data Space).

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

LDES member: object whose status cannot change after it was created

LDES view: event stream view that connects the collection to the root node such that all members of the collection can be found

Linked Data Event Stream (LDES): collection of immutable objects (such as version objects & sensor observations)

NGSI-LD attribute: reference to both an NGSI-LD Property and to an NGSI-LD Relationship

NGSI-LD attribute Type: categorization of an NGSI-LD Property or NGSI-LD Relationships

NGSI-LD element: any JSON element that is defined by the NGSI-LD API

NGSI-LD entity: informational representative of something that is supposed to exist in the real world, physically or conceptually

NGSI-LD entity type: categorization of an NGSI-LD entity as belonging to a class of similar entities, or sharing a set of characteristic properties

TREE collection: collection containing members following an imposed shape

NOTE: The members may be spread across multiple nodes.

TREE member: object that is part of a collection

TREE node: HTTP page, which itself may already contain members, that can contain relationships to other nodes

TREE ontology: hypermedia specification for fragmenting collections

TREE root node: home page, which itself may already contain members, through whose linked pages all members can be found

TREE shape: technical description of the structure to which objects from the collection has to conform

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
DCAT	Data CATalog vocabulary
FSDS	Flemish Smart Data Space
HTTP	HperText Transfer Protocol
JSON	Java Script Object Notation
LDES	Linked Data Event Streams
NGSI	Next Generation Service Interface
OGC	Open Geospatial Consortium
SEMIC	Semantic Interoperability Community
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WFS	Web Feature Service

NOTE: TREE is not an abbreviation, but the name of an ontology for fragmenting collections of items across multiple HTTP pages in a 'tree'-like way (linear, hierarchical, tiling-based, etc.).

4 Linked Data Event Streams

4.1 Introduction

The Linked Data Event Streams (LDES) specification enables datasets to be repurposed in different contexts. All kinds of datasets are in scope: static data, such as bibliographic records, as well as real-time data, such as sensor observations. A dataset is represented as an event source, which is an append-only collection of immutable objects: each time the dataset changes, a new object, called member, is added to the collection and is expected never to change. Typically, a member is a version object, but this can also be an immutable sensor observation or an activity object describing the occurred action.

The LDES specification [i.1] is available at https://w3id.org/ldes/specification.

NOTE 1: A collection imposes a certain shape on its members expressing the structure of the members. For example, members require to have a type "Vehicle" and a property "speed".

EXAMPLE 1: "Bob's vehicle speed is 10 km/h" can be modelled as an immutable sensor observation.

EXAMPLE 2: "Bob's vehicle brand is Mercedes" can be modelled as a version object of Bob's vehicle.

EXAMPLE 3: "Bob's vehicle's speed was measured at 10, 20, and 30 km/h" can be represented with three LDES members.

The LDES specification is based on the TREE hypermedia specification to publish collections. When a collection becomes too big for one HTTP page, fragmentation in multiple pages is needed. The TREE specification originates from the idea to provide an alternative to one-dimensional HTTP pagination. It allows to fragment a collection of items and interlink these fragments. Instead of linking to the next or previous page, a relation between two fragments, called TREE nodes, describes what elements can be found by following the link to another fragment.

The TREE specification [i.2] is available at https://w3id.org/tree/specification.

EXAMPLE 4: A TREE node identified with query parameter "time=2024-05-14" contains members with a modifiedAt property greater than "2024-05-14".

EXAMPLE 5: A TREE relation describes that TREE node <?time=2024-05-14> has members with a modified greater than "2024-05-14". The relation has as specific type "tree:GreaterThanRelation", has tree:node property with value <?time=2024-05-14>, and has a tree:path property with value modifiedAt.

NOTE 2: LDES typically uses one-dimensional pagination where updates are appended to a "latest" fragment. This way, consumers only need to be subscribed to one fragment for updates. When this latest fragment reaches a certain threshold, a new, latest fragment is created.

NOTE 3: The members of an LDES can be published through different types of fragmentations, which are called TREE views. For example, a view can first fragment a collection in geographical regions, and then enable a one-dimensional pagination per region.

With LDES, data consumers can set up pipelines to automatically replicate the history of a dataset and stay in sync with the latest updates. Today, custom code has to be created to integrate data, which makes it rather expensive to integrate multiple data sources. With LDES, a technical standard [i.1]was created that allows data to be exchanged across silos using domain-specific ontologies. An LDES allows third parties to build specific services (ETSI GS CIM 009 [i.3], Web Feature Service (WFS) [i.4], OGC APIs [i.5], SPARQL [i.6]) that are always in sync with the original dataset.

The next clauses describe how the TREE and LDES models map with the NGSI-LD meta model.

4.2 TREE to NGSI-LD mapping

The TREE vocabulary defines 5 classes, and 16 properties in total. The present document only maps the classes and properties that are required to be retrievable by LDES clients. Features like imports, view descriptions, shapes, remaining items, etc. are optional in the TREE specification [i.2] and therefore out-of-scope. Figure 4.2-1 demonstrates for a subset of TREE classes and properties how they map to NGSI-LD Entities, Relationships, Properties and value.

NOTE 1: Following prefixes are used to enhance readability:

- tree: https://w3id.org/tree#
- dcat: http://www.w3.org/ns/dcat#

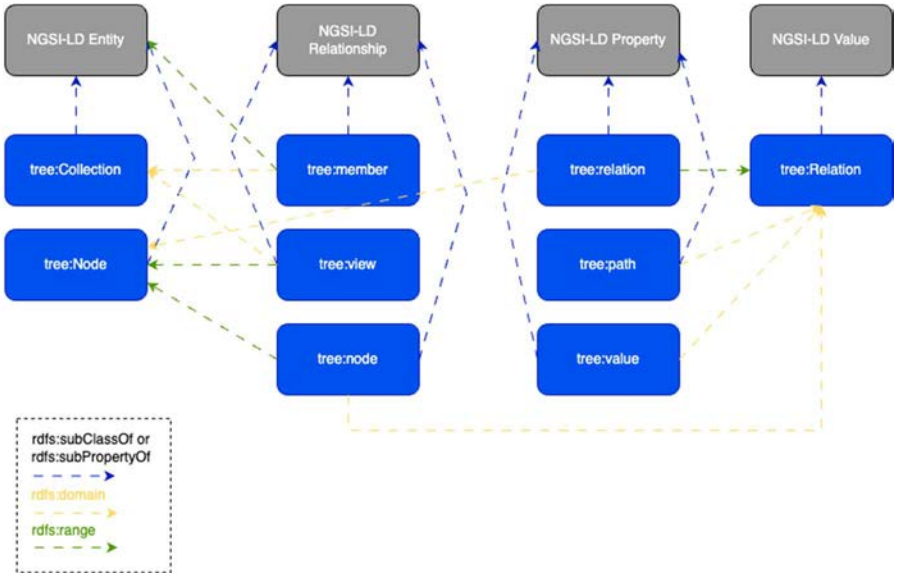

Figure 4.2-1: High-level mapping of TREE Specification to NGSI-LD Meta Model

The class tree:Collection can be mapped to NGSI-LD entity. A collection corresponds with a dcat:Dataset and has two relationships:

- The tree:member relationship linking to other members (NGSI-LD entities) that are part of the TREE collection. Clause A.1 provides an example.
- The tree:view relationship towards a tree:Node being an entrypoint (HTTP page) to retrieve members. Clause A.2 provides an example.

NOTE 2: The tree:member relationship typically only lists the members that are described in the same page.

The class tree:Node, being an HTTP page containing a part of the collection, can also be mapped to an NGSI-LD entity and has following property:

- the tree:relation property linking to another tree:Node. Clause A.2 provides an example.

The value of the tree:relation is an instance of a tree:Relation and is mapped as an NGSI-LD Value. More specifically, as a JSON-LD structured value. The reason is that a tree:Relation always is part of a tree:Node and therefore has no identity. The tree:Relation has following properties:

- A tree:node property (mandatory) containing the URL of another TREE node.
- A tree:path property (optional) indicating to which of the members' properties this relation applies.
- A tree:value property (optional) indicating a value constraint on the members' values.

NOTE 3: The tree:path and tree:value properties have to be used in conjunction to be useful for an LDES consumer.

NOTE 4: The tree:path and tree:value properties allow LDES consumers to decide whether to continue to the next node.

This clause demonstrates how TREE allows to:

i) define a collection of NGSI-LD Entities;

ii) define nodes that contain a subset of the collection; and

iii) use qualified relations to guide consumers to retrieve relevant nodes.

In clause 4.3, it is presented how LDES is a specialization of a TREE collection to replicate versions of entities.

4.3 LDES to NGSI-LD mapping

The LDES vocabulary defines 7 classes, and 8 properties in total. Features like event source, retention policies, version or materialization are not required to be retrievable by LDES clients and therefore out-of-scope.

Figure 4.3-1 shows that the ldes:EventStream class is mapped to an NGSI-LD entity, similar to its super class tree:Collection. Two properties are introduced to enable versioning on the entities of a collection:

- The ldes:timestampPath is mapped to an NGSI-LD Property indicating how a member precedes another member in the LDES using a timestamp.
- The ldes:versionOfPath is mapped to an NGSI-LD Property indicating the non-version object of a member.

An example of these properties and how this resolves in an LDES member can be found in clauses 5.1 and A.3.

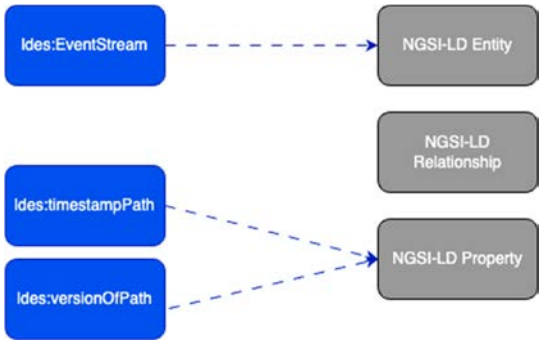

Figure 4.3-1: High-level mapping of Linked Data Event Streams data model to NGSI-LD Meta Model

This clause describes how LDES introduces versioning of entities using the timestamp and version of properties. This way, instances of the same entity have an explicit identifier and consumers know whether an entity version precedes another. In contrast with clause 4.6.6 in ETSI GS CIM 009 [i.3], entity instances of the same entity are not expected to come in chronological order. In clause 5, a way will be proposed to map an instance of an NGSI-LD entity to an LDES member.

5 Mapping NGSI-LD to LDES

5.0 Foreword

This clause 5 describes what needs to be adapted to the NGSI-LD Meta Model to map with the LDES standard. First, it will be discussed how the NGSI-LD entity representation from the core interfaces maps to one member. Then, it will be presented how the temporal representation of an entity can be mapped to multiple members.

5.1 NGSI-LD Data Representation to LDES mapping

5.1.1 NGSI-LD entity representation to LDES member

Here is a presentation of an instance of an entity with identifier urn:person:1. The system attributes createdAt and modifiedAt are provided to explain how a version can be created based on these properties.

EXAMPLE 1: NGSI-LD entity as base example to demonstrate LDES members:

```
{
   "id": "urn:person:1",
   "type": "Person",
   "name": {
     "type": "Property",
     "value": "Bill Smith",
     "createdAt": "2022-08-09T18:25:02Z",
     "modifiedAt": "2022-08-10T18:25:02Z"
   },
   "createdAt": "2022-08-09T18:25:02Z",
   "modifiedAt": "2022-08-10T18:25:02Z"
}
```

A member of an LDES represents an entity at a certain time. With the modifiedAt temporal property of an NGSI-LD entity, the last time the entity has been modified can be used to describe the latest version of an entity. Example 2 shows how the modifiedAt value is appended to create a version-specific identifier. For the timestamp property of a member, it is not allowed to use modifiedAt, because this is a system attribute and is preserved for internal usage in a context broker. Therefore, the present document proposes to define the observedAt temporal property at the entity level with following definition:

- observedAt is defined as the temporal Property at which a certain Entity became valid or was observed. For example, a person Entity with certain Properties and Relationships was observed at this point in time.

Example 2 shows how the observedAt property was added to the entity version. This version represents how the entity was observed at the time of modification.

It is also proposed to add the observedAt property on all the Relationships and Properties where this is not yet defined. Example 1 shows how the observedAt property was added to the name Property.

An isVersionOf property is added as non-reified property (the value only contains the entity id) containing the non-version entity id. Typically, LDESs use isVersionOf from the Dublin Core initiative (http://purl.org/dc/terms/isVersionOf).

All system attributes are hidden as this should not be part of the LDES publication.

EXAMPLE 2: Versioned entity using the observedAt temporal property:

```
{
   "id": "urn:ngsi-ld:Person:1:2022-08-10T18:25:02Z",
   "type": "Person",
   "isVersionOf": "urn:ngsi-ld:Person:1",
   "observedAt": "2022-08-10T18:25:02Z",
   "name": {
     "type": "Property",
     "value": "Bill Smith",
     "observedAt": "2022-08-10T18:25:02Z"
   }
}
```

EXAMPLE 3: A simplified representation of the versioned entity is also possible:

```
{
   "id": "urn:ngsi-ld:Person:1:2022-08-10T18:25:02Z",
   "type": "Person",
   "isVersionOf": "urn:ngsi-ld:Person:1",
   "observedAt": "2022-08-10T18:25:02Z",
   "name": "Bill Smith"
}
```

EXAMPLE 4: When observedAt is already defined, then this is not replaced with its modifiedAt value:

```
{
   "id": "urn:ngsi-ld:Person:1:2022-08-10T18:25:02Z",
   "type": "Person",
   "isVersionOf": "urn:ngsi-ld:Person:1",
   "observedAt": "2022-08-10T18:25:02Z",
   "name": {
     "type": "Property",
     "value": "Bill Smith",
     "observedAt": "1990-01-01T00:20:02Z"
   }
}
```

EXAMPLE 5: An HTTP(S) URI can also be used:

```
{
   "id": "https://example.org/id/person/1/2022-08-10T18:25:02Z",
   "type": "Person",
   "isVersionOf": "https://example.org/id/person/1",
   "observedAt": "2022-08-10T18:25:02Z",
   "name": "Bill Smith"
}
```

5.1.2 Temporal Representation of an Entity to LDES member

The temporal representation of an entity extends the NGSI-LD entity representation by composing each Property and Relationship with the sequence of instances of the referred Property during a period of time within its lifetime. Example 1 demonstrates how the name of a person is changed from "Joe Bloggs" to "Bill Smith".

EXAMPLE 1: Base example of an NGSI-LD entity using the temporal representation:

```
{
   "id": "urn:ngsi-ld:Person:1",
   "type": "Person",
   "name": {
     "type": "Property",
     "values": [
       {
            "type": "Property",
            "value": "Joe Bloggs",
            "instanceId": "instanceid:fb6b8ca8-d563-42ae-9a67-8b5cc97dc3ee",
          "createdAt": "2022-08-09T18:25:02Z",
            "modifiedAt": "2022-08-09T18:25:02Z"
       },
       {
            "type": "Property",
            "value": "Bill Smith",
            "instanceId": "instanceid: 6deb1f47-280c-4b18-866f-f49ea2f9cd7f",
            "createdAt": "2022-08-10T18:25:02Z",
            "modifiedAt": "2022-08-10T18:25:02Z"
         }
     ]
   },
   "createdAt": "2022-08-09T18:25:02Z",
   "modifiedAt": "2022-08-10T18:25:02Z"
}
```

To convert the temporal representation of Example 1 to LDES, the evolution in time of the entity needs to be reconstructed with the createdAt and modifiedAt properties. From these properties, one member represents the entity at creation time with name "Joe Bloggs". Another member represents the entity when it was modified with the name "Bill Smith". Example 2 gives an overview of these members.

EXAMPLE 2: Conversion of the base example with NGSI-LD temporal representation to two LDES members:

```
[{
      "id": "urn:ngsi-ld:Person:1:2022-08-09T18:25:02Z",
      "type": "Person",
      "isVersionOf": "urn:ngsi-ld:Person:1",
      "observedAt": "2022-08-09T18:25:02Z",
      "name": {
        "type": "Property",
        "value": "Joe Bloggs",
        "observedAt": "2022-08-09T18:25:02Z"
```

```
      }
}, {

      "id": "urn:ngsi-ld:Person:1:2022-08-10T18:25:02Z",
      "type": "Person",
      "isVersionOf": "urn:ngsi-ld:Person:1",
      "observedAt": "2022-08-10T18:25:02Z",
      "name": {
        "type": "Property",
        "value": "Bill Smith",
        "observedAt": "2022-08-10T18:25:02Z"
      }
   }
}]
```

6 Architectural Considerations

6.0 Foreword

This clause provides an overview how LDES sources that can be integrated into a context broker. Also, which fragmentations would be interesting for context brokers to expose LDESs will be discussed.

6.1 LDES to NGSI-LD

Figure 6.1-1 gives a general overview how the data from an LDES can be reused through a context broker. A context producer uses an LDES client to provision LDES members as context information to the context broker. The LDES client retrieves LDES members in an automated way by fetching TREE nodes, parsing members and following TREE relations towards next nodes. Each member is then sent to the NGSI-LD API of the context broker by the context producer.

Figure 6.1-1: Architectural overview of a context producer retrieving data from an LDES towards a NGSI-LD context broker

Following assumptions are made on the structure of the LDES members and how they can be sent to the NGSI-LD API:

ASSUMPTION 1: The LDES uses the NGSI-LD datamodel

LDES is a transport protocol for sharing a dataset and thus does not mandate which datamodel has to be used. It is assumed that the LDES contains NGSI-LD entities as members. When another datamodel is used, an extra transformation step will be needed by the context producer before sending the member to the NGSI-LD API, or a derived LDES has to be published where the transformation has already been done. How this transformation has to be done is out of scope of the present document.

ASSUMPTION 2: Members are versioned NGSI-LD Entities

The NGSI-LD Entities are assumed to be versioned to be retrievable by generic LDES clients. As NGSI-LD does not have the concept of versioning, it is proposed in clause 5 with a mapping from the NGSI-LD (Temporal) Entity representation to versioned LDES members. Versioning is enabled by using a dedicated version URI, and adding a versionOf Relationship and timestamp Property.

ASSUMPTION 3: Context producer materializes the members to non-versioned NGSI-LD Entities

Before sending the LDES member, being a versioned NGSI-LD entity, to the NGSI-LD API, a version materialization step should be performed by the Context Producer, because NGSI-LD systems do not have the concept of versioning. Otherwise, the Context Broker would treat every version Entity as a separate Entity. Materialization implies that the "isVersionOf" property is removed and the non-version identifier is used. Applying materialization to example 2 of clause 5.1.2 leads to following entities:

EXAMPLE 1: Materialization of versioned NGSI-LD Entities:

```
[{
      "id": "urn:ngsi-ld:Person:1",
      "type": "Person",
      "observedAt": "2022-08-09T18:25:02Z",
      "name": {
        "type": "Property",
        "value": "Joe Bloggs",
        "observedAt": "2022-08-09T18:25:02Z"
      }
}, {

      "id": "urn:ngsi-ld:Person:1",
      "type": "Person",
      "observedAt": "2022-08-10T18:25:02Z",
      "name": {
        "type": "Property",
        "value": "Bill Smith",
        "observedAt": "2022-08-10T18:25:02Z"
      }
   },
}]
```

ASSUMPTION 4: Members can be out-of-order and historical

Depending on the LDES view, members can be fragmented in different ways (geospatial, time-based, etc.). The members that are returned by the LDES client can be out-of-order and older than what is already available in the context broker. An assumption is that the context broker relies on the context producer doing the right thing: the core API should be used when the member is new or more recent than an existing Entity, else the temporal API should be used.

NOTE 1: Batching could be used on both the core and temporal API to provide multiple members with one HTTP request. With the core API, sorting is required where the last entity is the latest version. With the temporal API, entities may be provided unordered.

The following algorithm could be used:

- Create the Entity if it does not exist (i.e. no Entity with the same Entity Id is present).
- If there were an existing Entity with the same Entity Id, an update mechanism should be applied:
  - If the observedAt of the member is more recent than the observedAt property of the existing entity, the entity will be replaced or updated depending on the update mode using the Core API.
  - Otherwise, the entity will be treated as historical and should be sent to the temporal API of the context broker. A Temporal Representation has to be created from the member before sending to the temporal API. Example 2 (see below) provides an example of converting the member towards the temporal representation. Depending on the assumptions, there may be the need to check whether the historical instance (property or relationship) already exists to avoid duplication.
  - Optionally, each property and relationship of the member should be checked whether its observedAt timestamp is more recent than the observedAt timestamp of the property or relationship in the existing entity:
    - Create the Property or Relationship if it does not exist using the Core API.
    - Update the Property or Relationship if the property or relationship of the member is more recent, using the Core API.

- Otherwise, the property or relationship will be treated as historical and should be sent to the temporal API of the context broker. A Temporal Representation has to be created from the member before sending to the temporal endpoint.

NOTE 2: The context producer should probably remove the "observedAt" at the Entity level. This property should be generated automatically by the context broker.

EXAMPLE 2: Temporal Representation generated from the versioned NGSI-LD Entities of example 1:

```
[{
      "id": "urn:ngsi-ld:Person:1",
      "type": "Person",
      "name": {
         "type": "Property",
         "values": [{
            "type": "Property",
            "value": "Joe Bloggs",
            "observedAt": "2022-08-09T18:25:02Z"
         }, {
            "type": "Property",
            "value": "Bill Smith",
            "observedAt": "2022-08-10T18:25:02Z"
         }]
      }
}]
```

6.2 NGSI-LD to LDES

6.2.0 Foreword

The first step of publishing an LDES is versioning a dataset with immutable members. How this can be done with NGSI-LD Entities has been discussed in clause 5.1. Then, one or more views needs to be configured how these members have to be published.

Clause 6.2 first describes the algorithm to create a paginated view. Then, it is demonstrated how this view can be extended with fragmentations, such as geospatial, timebased and reference-based fragmentations.

NOTE: How the LDES views are structured is up to the publisher. Clause 6.2 shows how views are currently tackled within the LDES Server of the Flemish Smart Data Space project [i.7].

6.2.1 Paginated view

A pagination needs to be created in which an initial set of members can be found through the first tree:Node, and more members can be found by interpreting the TREE hypermedia controls.

When the size of an LDES collection becomes too big for 1 HTTP response, pagination will be required in which an initial set of members can be found through the first tree:Node, and more members can be found by interpreting the TREE hypermedia controls [i.2]. One approach is to use partitioning, which means that the LDES server will create fragments based on the order of arrival of the LDES members. The latest arriving members are appended to the "latest page". When this latest page reaches a certain size, this page becomes immutable and a new latest page will be created. This approach is efficient for Open Data publishing as it allows to set a max-age: immutable header on the pages that will not be updated anymore. The algorithm for this append-only, paginated view [i.10] works as follows:

- The fragment to which the member should be added is determined:
  - The currently open fragment is retrieved from the database.
  - If this fragment contains members equal to or exceeding the member limit or no fragment can be found, a new fragment is created instead.
- If a new fragment is created, the following steps are taken:
  - The new fragment becomes the new open fragment and the previous fragment becomes immutable.

- This newly created fragment and the previous fragment are then linked with each other by 2 generic relationships.
- The page number of the new fragment is determined based on the old fragment or is set to 1 in case of the first fragment.

Example 1 shows how the configuration of a view can be done using a separate tree:ViewDescription object. This ViewDescription allows alignment with DCAT, but also to describe fragmentation strategies and retention policies (out of scope of the present document). tree:viewDescription is mapped as an NGSI-LD Relationship, because this description could be referenced from several places, such as DCAT catalogs or TREE nodes that share the same view description.

EXAMPLE 1: A tree:ViewDescription is attached to a view to indicate that a page size of 100 members is configured. For demonstration purposes, an empty list for the fragmentation strategy and retention policy was added. A relation is added to the first page.

```
{
    "id": "/by-page",
    "type": "tree:Node",
    "tree:viewDescription": {
         "type": "Relationship",
         "value": {
            "type": "tree:ViewDescription",
            "tree:fragmentationStrategy": [],
            "tree:retentionPolicy": [],
            "tree:pageSize": 100
    },
     "tree:relation": [{
       "type": "Property",
       "value": {
         "type": "tree:Relation",
         "tree:node": "/by-page?page=1"
       }
    }]
}
```

NOTE: A JSON-LD context of LDES can be found on Github [i.8].

6.2.2 Geospatial fragmentation

Views can be configured with a geospatial fragmentation strategy. Example 1 shows how a tiling-based approach [i.9] can be configured on a view. The property tree:maxZoom allows defining the zoom level of the tiles that will be generated. The property tree:fragmentationPath allows defining the location property that will be used to fragment the member in tiles.

EXAMPLE 1: The view description indicates that a geospatial fragmentation strategy is used on the ngsi-ld:location property and tiles at zoom level of 14 will be created.

```
{
    "id": "/by-location",
    "type": "tree:Node",
    "tree:viewDescription": {
         "type": "Relationship",
         "value": {
            "type": "tree:ViewDescription",
            "tree:fragmentationStrategy": [{
               "type": "tree:GeospatialFragmentation",
               "tree:maxZoom": "14",
               "tree:fragmentationPath": "ngsi-ld:location"
            }]
         }
    }
}
```

To traverse from the /by-location view to the different tiles, a root tile 0/0/0 can be used as intermediate node. When the number of tiles becomes too much to list in one page, pagination on the root tile can be performed.

EXAMPLE 2: The /by-location view links to a root tile.

```
{
    "id": "/by-location",
    "type": "tree:Node",
    "tree:relation": {
      "type": "Property",
      "value": {
         "type": "tree:Relation",
         "tree:node": "/by-location?tile=0%2F0%2F0"
    }
}
```

The TREE specification provides specializations of tree:Relations, such as tree:GeospatiallyContainsRelation to indicate a geospatial constraint on the members within a node.

EXAMPLE 3: The root tile contains a list of GeospatiallyContains relations to the tiles on the zoom level. In this example, the tile 14/7762/5275 only contains members that have a ngsi-ld:location within the GeoJSON polygon.

```
{
    "id": "/by-location?tile=0%2F0%2F0",
    "type": "tree:Node",
    "tree:relation": [{
       "type": "Property",
       "value": {
         "type": "tree:GeospatiallyContainsRelation",
         "tree:node": "/by-location?tile=14%2F7762%2F5275",
           "tree:path": "ngsi-ld:location",
           "tree:value": {
                  "value": {
                    "type": "Polygon",
                    "coordinates": [
                      [
                        [-9.42626953125, 53.813625792352354],
                        [-9.42626953125, 53.80065082633022],
                        [-9.4482421875, 53.80065082633022],
                        [-9.4482421875, 53.813625792352354],
                        [-9.42626953125, 53.813625792352354]
                      ]
                    ]
                  }
            }
       }
    }]
  }
```

Finally, pagination should be applied on the specific tile when too many members are added within that region. More information on the algorithm can be found here: https://informatievlaanderen.github.io/VSDS-LDESServer4J/3.5.0-SNAPSHOT/configuration/fragmentations/geospatial [i.12].

6.2.3 Reference fragmentation

Reference fragmentation is used to filter members that have a relationship to a certain entity. Example 1 shows how Person 1 is a friend of Person 2. Example 2 shows how the configuration of a reference fragmentation uses tree:fragmentationPath to fetch the value on which the member should be bucketized, and tree:fragmentationKey to provide a query parameter to search on.

Applying example 1 with the fragmentation of example 2 generates a fragment with URL: /by-friendOf?friendOf=urn:ngsi-ld:Person:2. This fragment only contains members that have a friendOf relationship with Person:2.

EXAMPLE 1: Person:1 is a friend of Person:2, using a relationship as example.

```
{
      "id": "urn:ngsi-ld:Person:1:2022-08-10T18:25:02Z",
      "type": "Person",
      "isVersionOf": "urn:ngsi-ld:Person:1",
      "observedAt": "2022-08-10T18:25:02Z",
      "example:friendOf": {
         "type": "Relationship",
        "object": "urn:ngsi-ld:Person:2"
```

```
      }
}
```

EXAMPLE 2: A reference fragmentation provides a query parameter using tree:fragmentationKey to allow filtering members on a certain relationship, which is defined with tree:fragmentationPath.

```
{
    "id": "/by-friendOf",
    "type": "tree:Node",
    "tree:viewDescription": {
         "type": "Relationship",
         "value": {
            "type": "tree:ViewDescription",
            "tree:fragmentationStrategy": [{
               "type": "tree:ReferenceFragmentation",
               "tree:fragmentationKey": "friendOf",
               "tree:fragmentationPath": "example:friendOf"
            }]
         }
    }
}
```

More information about this fragmentation strategy can be found on the FSDS Web page [i.11].

Annex A:
Mapping examples

A.1 Mapping TREE collection and its members

An example TREE collection is represented below in Turtle format. The tree:Collection is mapped to an NGSI-LD entity and has a NGSI-LD relationship towards its members, which in this case are NGSI-LD Entities.

```
@prefix ex: <http:example.org/> .
@prefix tree: <https://w3id.org/tree#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ngsi-ld: <https://uri.etsi.org/ngsi-ld/> .
@prefix sdm: <https://smartdatamodels.org/> .
@prefix sdm-o: <https://smartdatamodels.org/datamodel.Organization/> .

ex:Collection1 a tree:Collection;
            dterms:description [ a ngsi-ld:LanguageProperty ;
                          ngsi-ld:hasLanguageMap "A Collection of people."@en
                        ] ;
            tree:member <urn:ngsi-ld:Person:Alice>, <urn:ngsi-ld:Person:Bob> .

<urn:ngsi-ld:Person:Alice> a sdm-o:Person;
            sdm:givenName [ a ngsi-ld:Property ;
                   ngsi-ld:hasValue "Alice"
            ] .

<urn:ngsi-ld:Person:Bob> a sdm-o:Person;
            sdm:givenName [ a ngsi-ld:Property ;
                   ngsi-ld:hasValue "Bob"
            ] .
```

For this example, the NGSI-LD entities generated are displayed below in the NGSI-LD normalized format:

```
[
  {
    "id": "ex:Collection1",
    "type": "tree:Collection",
    "description": {
      "type": "LanguageProperty",
      "languageMap": {
         "en": "A Collection of people."
      }
    },
    "tree:member": {
      "type": "Relationship",
      "object": [
          "urn:ngsi-ld:Person:Alice",
          "urn:ngsi-ld:Person:Bob"
      ]
    }
  },
  {
    "id": "urn:ngsi-ld:Person:Alice",
    "type": "Person",
    "givenName": {
      "type": "Property",
      "value": "Alice"
    }
  },
  {
    "id": "urn:ngsi-ld:Person:Bob",
    "type": "Person",
    "givenName ": {
      "type": "Property",
      "value": "Bob"
    }
  }
]
```

And the @context vocabulary would be:

```
{
  "@context": [
    {
      "ex": "http://example.org/",
      "tree": "https://w3id.org/tree#"
    },
    "https://raw.githubusercontent.com/smart-data-
models/dataModel.Organization/master/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.8.jsonld"
  ]
}
```

A.2 Mapping TREE collection and its view

An example view of a TREE collection is represented below. The tree:Collection Entity has a tree:view NGSI-LD relation to a tree:Node Entity. A tree:Node that is referred to as view acts as an entry node to retrieve all members (not all nodes allow a client to retrieve all members). A tree:Node has a tree:relation NGSI-LD property describing the relation towards another node. In the example, a simple (tree:Relation) and a complex (tree:EqualToRelation) relation are demonstrated.

```
@prefix ex: <http:example.org/> .
@prefix tree: <https://w3id.org/tree#> .
@prefix ngsi-ld: <https://uri.etsi.org/ngsi-ld/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix sdm: <https://smartdatamodels.org/> .

ex:Collection1 a tree:Collection;
            dcterms:description[ a ngsi-ld:LanguageProperty ;
                         ngsi-ld:hasLanguageMap "A Collection of NGSI-LD Entities"@en
                        ] ;
            tree:view </?page=1> .


</?page=1> a tree:Node;
            dcterms:description [ a ngsi-ld:Property ;
                   ngsi-ld:hasValue "First page"
            ] ;
             tree:relation [
                   a tree:Relation ;
                   dcterms:description [ a ngsi-ld:Property ;
                       ngsi-ld:hasValue "Simple relation that refers to a next page"
                   ] ;
                   tree:node </?page=2>
            ] , [
                   a tree:EqualToRelation ;
                   dcterms:description [ a ngsi-ld:Property ;
                       ngsi-ld:hasValue "Second, complex relation that refers to a page containing
members with an address locality equal to Berlin."
                   ] ;
                   tree:node </?addressLocality=Berlin> ;
                   tree:value "Berlin" ;
                   tree:path (sdm:address sdm:addressLocality)
            ] .
```

For this example, the NGSI-LD entities generated are displayed below in the NGSI-LD normalized format:


```
[
  {
    "id": "ex:Collection1",
    "type": "tree:Collection",
    "description": {
      "type": "LanguageProperty",
      "languageMap": {
         "en": "A Collection of NGSI-LD Entities"
      }
    },
    "tree:view": {
      "type": "Relationship",
      "object": {
          "id": "/?page=1"
      }
    }
  },
  {
    "id": "/?page=1",
    "type": "tree:Node",
    "description": {
      "type": "Property",
      "value": "First page"
    },
    "tree:relation": [{
       "type": "Property",
       "value": {
         "type": "tree:Relation",
         "description": "Simple relation that refers to a next page.",
         "tree:node": {
             "id": "/?page=2"
          }
       }
    }, {
       "type": "Property",
       "value": {
         "type": "tree:EqualToRelation",
         "description": "Second, complex relation that refers to a page containing members with an
address locality equal to Berlin.",
         "tree:node": "/?addressLocality=Berlin",
         "tree:value": "Berlin",
         "tree:path": {
             "type": "ListProperty",
            "valueList": [ {
                   "id": "sdm:address"
            }, {
                   "id": "sdm:addressLocality"
            }]
         }
       }
    }
    ]
  }
]
```


And the @context vocabulary would be:


```
{
  "@context": [
    {
      "ex": "http://example.org/",
      "sdm": "https://smartdatamodels.org/",
      "tree": "https://w3id.org/tree#",
      "tree:node": {"@type": "@id"}
    },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.8.jsonld"
  ]
}
```

A.3 Mapping LDES collection and its members

An example LDES collection is represented below.

```
@prefix ex: <http:example.org/> .
@prefix tree: <https://w3id.org/tree#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ngsi-ld: <https://uri.etsi.org/ngsi-ld/> .
@prefix sdm: <https://smartdatamodels.org/> .
@prefix sdm-o: <https://smartdatamodels.org/datamodel.Organization/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:LdesCollection1 a ldes:EventStream;
          dcterms:description [ a ngsi-ld:LanguageProperty ;
                          ngsi-ld:hasLanguageMap "An LDES Collection of NGSI-LD Entities"@en
                        ] ;
          ldes:timestampPath ngsi-ld:observedAt ;
          ldes:versionOfPath dcterms:isVersionOf ;
          tree:member <urn:ngsi-ld:Person:Alice:2022-08-10T18:25:02Z>, <urn:ngsi-ld:Person:Bob:2022-
08-10T18:25:02Z> .

<urn:ngsi-ld:Person:Alice:2022-08-10T18:25:02Z> a sdm-o:Person;
            sdm:givenName [ a ngsi-ld:Property ;
                   ngsi-ld:hasValue "Alice";
                   ngsi-ld:observedAt "2022-08-10T18:25:02Z"^^ngsi-ld:DateTime
            ] ;
            dcterms:isVersionOf <urn:ngsi-ld:Person:Alice> ;
            ngsi-ld:observedAt "2022-08-10T18:25:02Z"^^ngsi-ld:DateTime .

<urn:ngsi-ld:Person:Bob:2022-08-10T18:25:02Z> a sdm-o:Person;
            sdm:givenName [ a ngsi-ld:Property ;
                   ngsi-ld:hasValue "Bob" ;
                   ngsi-ld:observedAt "2022-08-10T18:25:02Z"^^ngsi-ld:DateTime
            ] ;
          dcterms:isVersionOf <urn:ngsi-ld:Person:Bob> ;
          ngsi-ld:observedAt "2022-08-10T18:25:02Z"^^ngsi-ld:DateTime .
```

For this example, the NGSI-LD entities generated are displayed below in the NGSI-LD normalized format:

```
[
  {
    "id": "ex:Collection1",
    "type": "ldes:EventStream",
    "description": {
      "type": "LanguageProperty",
      "languageMap": {
         "en": "An LDES Collection of NGSI-LD Entities"
      }
    },
    "ldes:timestampPath": {
      "type": "Property",
      "value": "ngsi-ld:observedAt"
    },
    "ldes:versionOfPath": {
      "type": "Property",
      "value": "dcterms:isVersionOf"
    },
    "tree:member": {
      "type": "Relationship",
      "object": [
         "urn:ngsi-ld:Person:Alice:2022-08-10T18:25:02Z",
         "urn:ngsi-ld:Person:Bob:2022-08-10T18:25:02Z"
      ]
    }
  },
  {
    "id": "urn:ngsi-ld:Person:Alice:2022-08-10T18:25:02Z",
    "type": "Person",
    "givenName": {
      "type": "Property",
      "value": "Alice",
      "observedAt": "2022-08-10T18:25:02Z"
    },
    "dcterms:isVersionOf": "urn:ngsi-ld:Person:Alice",
```

```
    "observedAt": "2022-08-10T18:25:02Z"
  },
  {
    "id": "urn:ngsi-ld:Person:Bob:2022-08-10T18:25:02Z",
    "type": "Person",
    "givenName": {
      "type": "Property",
      "value": "Bob",
      "observedAt": "2022-08-10T18:25:02Z"
    },
    "observedAt": "2022-08-10T18:25:02Z"
  }
]
```

And the @context vocabulary would be:

```
{
  "@context": [
    {
      "ex": "http://example.org/",
      "tree": "https://w3id.org/tree",
      "ldes": "https://w3id.org/ldes#",
      "dcterms": "http://purl.org/dc/terms/",
      "dcterms:isVersionOf": {"@type": "@id"},
      "members": {"@type": "@id", "@id": "tree:member", "@container":"@list"},
      "timestampPath": {"@id":"ldes:timestampPath", "@type":"@id", "@container":"@list"},
      "versionOfPath": {"@id":"ldes:versionOfPath", "@type":"@id", "@container":"@list"}
    },

    "https://raw.githubusercontent.com/smart-data-
models/dataModel.Organization/master/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.8.jsonldh"
  ]
}
```

Annex B:
Change history

Date	Version	Information about changes
May 14th 2024	0.0.1	First draft
June 3rd 2024	0.0.2	Stable early draft
September 2024	0.0.4	Add mapping LDES/TREE to NGSI-LD. Expand representation mapping. Add architecture proposal
September 2024	0.0.5	Process feedback of weekly call
October 2024	0.0.6	Process feedback plenary (8th October 2024). Extend fragmentations NGSI-LD to LDES.
December 2024	0.0.7	Process feedback API call 11th December 2024. Add JSON-LD context of LDES
December 18th 2024	0.0.8-r1	Incorporate review NEC. Check formatting
January 2025	1.1.1	ETSI Technical Officer review before ETSI EditHelp publication preprocessing after TB approval

History

Document history		
V1.1.1	February 2025	Publication