ETSI GS CIM 009 V1.2.2 (2020-02)

Context Information Management (CIM); NGSI-LD API

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
RGS/CIM-0009v122

Keywords
API, architecture, GAP, information model,
interoperability, smart city, WoT

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 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is 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 Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, 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.

Foreword

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

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**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 formally describes the Context Information Management API (NGSI-LD) Specification. The Context Information Management API allows users to provide, consume and subscribe to context information in multiple scenarios and involving multiple stakeholders. It enables close to real-time access to information coming from many different sources (not only IoT data sources).

Introduction

The present document defines the Context Information Management API Specification. The Context Information Management API allows users to provide, consume and subscribe to context information in multiple scenarios and involving multiple stakeholders. The ongoing status of the NGSI-LD API can be found in [i.17].

The ETSI ISG CIM has decided to give the name "NGSI-LD" to the Context Information Management API. The rationale is to reinforce the fact that the present document leverages on the former OMA NGSI 9 and 10 interfaces [i.3] and FIWARE NGSIv2 [i.9] to incorporate the latest advances from Linked Data.

The present document provides additions and corrections to the ETSI GS CIM 004 [i.16] preliminary API specification, based on feedback about ETSI GS CIM 004 [i.16] received from other SDOs as well as developers in the linked-data, internet-of-things, and mobile-apps and smart-applications communities, as well as from end users and stakeholders. In particular, open issues and proposed features in annexes of the referred ETSI GS CIM 004 [i.16] document have been addressed or added respectively.

Most of the NGSI-LD API and the ETSI ISG CIM information model work referenced here was created with the support of the following European Union Horizon 2020 research projects: No. 732851 (FI-NEXT), No. 723156 (WISE-IoT), No. 732240 (SynchroniCity), No. 731993 (AutoPilot), No. 814918 (Fed4IoT), No. 779852 (IoTCrawler), and No. 731884 (IoF2020).

1 Scope

The purpose of the present document is the definition of a standard API for Context Information Management (NGSI-LD API) enabling close to real-time access to information coming from many different sources (not only IoT data sources). The present document defines how such an API enables applications to perform updates on context, register context providers which can be queried to get updates on context, query information on current and historic context information and subscribe to receive notifications of context changes. The criteria for choice of the API characteristics are based on requirements resulting from the Use Cases [i.1] and other work items [i.2] and [i.8].

The present document leverages on ETSI GS CIM 004 [i.16] prelimAPI specification, providing additions and corrections, based on feedback about ETSI GS CIM 004 [i.16] received from other SDOs as well as developers in the linked-data, internet-of-things, and mobile-apps and smart-applications communities, as well as from end users and stakeholders. In particular, it contains the resolution of some of the open issues and proposed features in annexes of ETSI GS CIM 004 [i.16].

2 References

2.1 Normative 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.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

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 necessary for the application of the present document.

[1] W3C Recommendation 25 February 2014: "RDF Schema 1.1".

NOTE: Available at https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[2] W3C Recommendation 16 January 2014: "JSON-LD 1.0 - A JSON-based Serialization for Linked Data".

NOTE: Available at http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[3] IETF RFC 7231: "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content".

NOTE: Available at https://tools.ietf.org/html/rfc7231.

[4] IETF RFC 7232: "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests".

NOTE: Available at https://tools.ietf.org/html/rfc7232.

[5] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

NOTE: Available at https://tools.ietf.org/html/rfc3986.

[6] IETF RFC 8259: "The JavaScript Object Notation (JSON) Data Interchange Format".

NOTE: Available at https://tools.ietf.org/html/rfc8259.

[7] IETF RFC 8288: "Web Linking".

NOTE: Available at https://tools.ietf.org/html/rfc8288.

[8] IETF RFC 7946: "The GeoJSON Format".

NOTE: Available at https://tools.ietf.org/html/rfc7946.

[9] IETF RFC 8141: "Uniform Resource Names (URNs)".

NOTE: Available at https://tools.ietf.org/html/rfc8141.

[10] IETF RFC 7807: "Problem Details for HTTP APIs".

NOTE: Available at https://tools.ietf.org/html/rfc7807.

[11] IEEE POSIX 1003.2™-1992: "IEEE Standard for Information Technology - Portable Operating System Interfaces (POSIX®) - Part 2: Shell and Utilities".

[12] IETF RFC 5234: "Augmented BNF for Syntax Specifications: ABNF".

NOTE: Available at https://tools.ietf.org/html/rfc5234.

[13] Unicode® Technical Standard #10: "Unicode Collation Algorithm".

NOTE: Available at http://unicode.org/reports/tr10/.

[14] Open Geospatial Consortium Inc. OGC 06-103r4: "OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture".

NOTE: Available at https://portal.opengeospatial.org/files/?artifact_id=25355.

[15] UN/CEFACT Common Codes for specifying the unit of measurement.

NOTE: Available at http://www.unece.org/fileadmin/DAM/cefact/recommendations/rec20/rec20_Rev9e_2014.xls.

[16] IETF RFC 7396: "JSON Merge Patch".

NOTE: Available at https://tools.ietf.org/html/rfc7396.

[17] ISO 8601: 2004: "Data elements and interchange formats -- Information interchange -- Representation of dates and times".

NOTE: Available at http://www.iso.org/iso/catalogue_detail?csnumber=40874.

[18] IETF RFC 2818: "HTTP Over TLS".

NOTE: Available at https://tools.ietf.org/html/rfc2818.

[19] IETF RFC 5246: "The Transport Layer Security (TLS) Protocol Version 1.2".

NOTE: Available at https://tools.ietf.org/html/rfc5246.

[20] IANA Registry of Link Relation Types.

NOTE: Available at https://www.iana.org/assignments/link-relations/.

[21] ECMA 262 Specification: "ECMAScript® 2018 Language Specification".

NOTE: Available at http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

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] ETSI GR CIM 002: "Context Information Management (CIM); Use Cases (UC)".

NOTE: Available at https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=51347.

[i.2] ETSI GR CIM 007: "Context Information Management (CIM); Security and Privacy".

NOTE: Available at https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=53370.

[i.3] OMA-TS-NGSI-Context-Management-V1-0-20120529-A: "NGSI Context Management".

NOTE: Available at http://www.openmobilealliance.org/release/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf.

[i.4] ETSI TS 103 264 (V2.1.1): "SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping".

[i.5] NGSI-LD Wrapper, Experimental proxy for adaptation between FIWARE and NGSI-LD.

NOTE: Available at https://github.com/Fiware/NGSI-LD_Wrapper.

[i.6] Graph Databases: "New Opportunities for Connected Data". O'Reilly 2nd Edition. Webber, Robinson, et al. ISBN:1491930896 9781491930892.

[i.7] JSON-LD Playground. Experimentation tool for JSON-LD.

NOTE: Available at https://json-ld.org/playground/.

[i.8] ETSI GS CIM 006: "Context Information Management (CIM); Information Model (MOD0)".

NOTE: Available at https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=51351.

[i.9] FIWARE NGSI REST binding version 2.

NOTE: Available at http://fiware.github.io/specifications/ngsiv2/stable/.

[i.10] IETF RFC 6902: "JavaScript Object Notation (JSON) Patch".

NOTE: Available at https://tools.ietf.org/html/rfc6902.

[i.11] JSON Schema Validation: "A Vocabulary for Structural Validation of JSON".

NOTE: Available at https://json-schema.org/latest/json-schema-validation.html.

[i.12] OpenAPI Specification (Swagger).

NOTE: Available at https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md.

[i.13] NGSI-LD JSON Schemas.

NOTE: Available at https://forge.etsi.org/gitlab/NGSI-LD/NGSI-LD/tree/master/schema.

[i.14] NGSI-LD OpenAPI (Swagger) Specification.

NOTE: Available at https://forge.etsi.org/gitlab/NGSI-LD/NGSI-LD/tree/master/spec.

[i.15] NGSI-LD Examples.

NOTE: Available at https://forge.etsi.org/gitlab/NGSI-LD/NGSI-LD/tree/master/examples.

[i.16] ETSI GS CIM 004: "Context Information Management (CIM); Application Programming Interface (API)".

[i.17] ETSI ISG CIM: "NGSI-LD Status".

NOTE: Available at https://docbox.etsi.org/ISG/CIM/Open/NGSI-LD Status.pdf.

[i.18] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

NOTE: The letters "NGSI-LD" were added to most terms to confirm that they are distinct from other terms of similar/same name in use in other organizations, however, in the present document the letters "NGSI-LD" are generally omitted for brevity.

NGSI-LD Attribute: reference to either the name of an NGSI-LD Property or to the name of an NGSI-LD Relationship

NGSI-LD Central Broker: NGSI-LD Context Broker that only uses a local storage when serving NGSI-LD requests, without involving any external Context Sources

NGSI-LD Context Broker: architectural component that implements all the NGSI-LD interfaces

NGSI-LD Context Consumer: agent that uses the query and subscription functionality of NGSI-LD to retrieve context information

NGSI-LD Context Producer: agent that uses the NGSI-LD context provision and/or registration functionality to provide or announce the availability of its context information to an NGSI-LD Context Broker

NGSI-LD Context Registry: software functional element where Context Sources register the information that they can provide

NOTE: It is used by Distribution Brokers and Federation Brokers to find the appropriate Context Sources which can provide the information required for serving an NGSI-LD request.

NGSI-LD Context Source: source of context information which implements the NGSI-LD consumption and subscription (and possibly provision) interfaces defined by the present document

NOTE: It is usually registered with an NGSI-LD Registry so that it can announce what kind of information it can provide, when requested, to Context Consumers and Brokers.

NGSI-LD Distribution Broker: NGSI-LD Context Broker that uses both local context information and registration information from an NGSI-LD Context Registry, to access matching context information from a set of distributed Context Sources

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

NOTE: In the NGSI-LD API, any instance of such an entity is **uniquely identified by a URI**, and characterized by reference to one or more **NGSI-LD Entity Type(s)**. The API defined by the present document only allows associating one NGSI-LD Entity Type per NGSI-LD Entity. This restriction will be removed in future versions.

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

NOTE: In the NGSI-LD API, an NGSI-LD Entity Type is **uniquely identified by a URI**.

EXAMPLE 1: "Vehicle" is an NGSI-LD Entity Type and is identified with a proper URI.

EXAMPLE 2: Bob's private car whose plate number is "ABCD1234" is an NGSI-LD Entity whose NGSI-LD Entity Type Name is "Vehicle".

NGSI-LD External Linked Entity: Linked Entity that is identified through a **dereferenceable URI** which does not exist within the current NGSI-LD system

NOTE: It can exist within another NGSI-LD system or within a non-NGSI-LD system.

EXAMPLE: An NGSI-LD Entity, which Entity Type Name is "Book", can be externally linked, through the "wasWrittenBy" relationship, to a resource identified by the URI "http://dbpedia.org/resource/Mark_Twain".

NGSI-LD Federation Broker: Distribution Broker that federates information from multiple underlying NGSI-LD Context Brokers and across domains

NGSI-LD Internal Linked Entity: Linked Entity that exists within the current NGSI-LD system

EXAMPLE: An NGSI-LD Entity, which Entity Type name is "Vehicle", can be internally linked, through the "isParkedAt" relationship, to another NGSI-LD Entity, of Type Name "Parking", identified by the URI "urn:ngsi-ld:Parking:Downtown1".

NGSI-LD Linked Entity: NGSI-LD Entity referenced from another NGSI-LD Entity (the linking NGSI-LD Entity) via an NGSI-LD Relationship

NGSI-LD Linking Entity: NGSI-LD Entity which is the subject of a Relationship to another NGSI-LD Entity (the linked NGSI-LD Entity) or an external resource (identified by a URI)

NGSI-LD Name: short-hand string (term) that locally identifies an NGSI-LD Entity Type, Property Type or Relationship Type and which can be mapped to a URI which serves as a fully qualified identifier

EXAMPLE: The sentence "Bob's vehicle's speed is 40 km/h" can be represented by an NGSI-LD Property, whose Name is "speed", and which characterizes an NGSI-LD Entity, which NGSI-LD Type Name is "Vehicle". Such a name can be expanded to a fully qualified name in the form of a URI, for instance "http://example.org/Vehicle" or "http://example.org/speed".

NGSI-LD Property: description instance which associates a main characteristic, i.e. an **NGSI-LD Value**, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property and that uses the special *hasValue* property to define its target value

NGSI-LD Relationship: description of a directed link between a subject which is either an NGSI-LD Entity, an NGSI-LD Property, or another NGSI-LD Relationship on one hand, and an object, which is an NGSI-LD Entity, on the other hand, and which uses the special *hasObject* property to define its target object

EXAMPLE: An NGSI-LD Entity of type (Type Name) "Vehicle" (when parked) can be the subject of an NGSI-LD Relationship which object is an NGSI-LD Entity of type "Parking".

NGSI-LD Value: JSON value (i.e. a string, a number, true or false, an object, an array), or a JSON-LD typed value (i.e. a string as the lexical form of the value together with a type, defined by an XSD base type or more generally an IRI), or a JSON-LD structured value (i.e. a set, a list, a language-tagged string)

EXAMPLE: Bob's private car 'speed' NGSI-LD Value is the number 100 (kilometres per hour).

3.2 Symbols

Void.

3.3 Abbreviations

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

ABNF	Augmented Backus-Naur Form
API	Application Programming Interface
BNF	Backus Naur Form
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IoT	Internet of Things
IRI	Internationalized Resource Identifier
ISG	Industry Specification Group
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JSON-LD	JSON Linked Data
NGSI	Next Generation Service Interfaces
NID	Namespace Identifier
NSS	Namespace Specific String
OAS	Open API Specification
OMA	Open Mobile Alliance
POSIX	Portable Operating System Interface
RDF	Resource Description Format
REST	Representational State Transfer
RFC	Request For Comments
SAREF	Smart Applications Reference ontology
TB	Technical Body
TLS	Transport Layer Security
UCA	Unicode Collation Algorithm
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Universal Resource Locator
URN	Uniform Resource Name
UTC	Coordinated Universal Time
UTF	Unicode (or Universal Coded Character Set) Transformation Format
XSD	XML Schema Definition

4 Context Information Management Framework

4.1 Introduction

This clause describes the technical design principles behind the context information management framework supported by NGSI-LD. As stated in clause 3.1, the letters "NGSI-LD" which are part of most terms, to confirm that they are distinct from other terms of similar/same name in use in other organizations, are generally omitted in the present document for brevity. In the present document, a number of rather obvious typographic conventions and syntax guidelines are followed and the reader is referred to annex F for details.

4.2 NGSI-LD Information Model

4.2.1 Introduction

The NGSI-LD Information Model prescribes the structure of context information that shall be supported by an NGSI-LD system. It specifies the data representation mechanisms that shall be used by the NGSI-LD API itself. In addition, it specifies the structure of the Context Information Management vocabularies to be used in conjunction with the API.

The NGSI-LD Information Model is defined at two levels (see figure 4.2.1-1): the foundation classes which correspond to the Core Meta-model and the Cross-Domain Ontology. The former amounts to a formal specification of the "property graph" model [i.6]. The latter is a set of generic, transversal classes which are aimed at avoiding conflicting or redundant definitions of the same classes in each of the domain-specific ontologies. Below these two levels, domain-specific ontologies or vocabularies can be devised. For instance, the SAREF Ontology ETSI TS 103 264 [i.4] can be mapped to the NGSI-LD Information Model, so that smart home applications will benefit from this Context Information Management API specification.

The version of the cross-domain model proposed by the present document is a minimal one, aimed at defining the classes used in this release of the API specification. It has been extended by other work items like ETSI GS CIM 006 [i.8], with classes defining extra concepts such as mobile vs. stationary entities, instantaneous vs. static properties, etc.

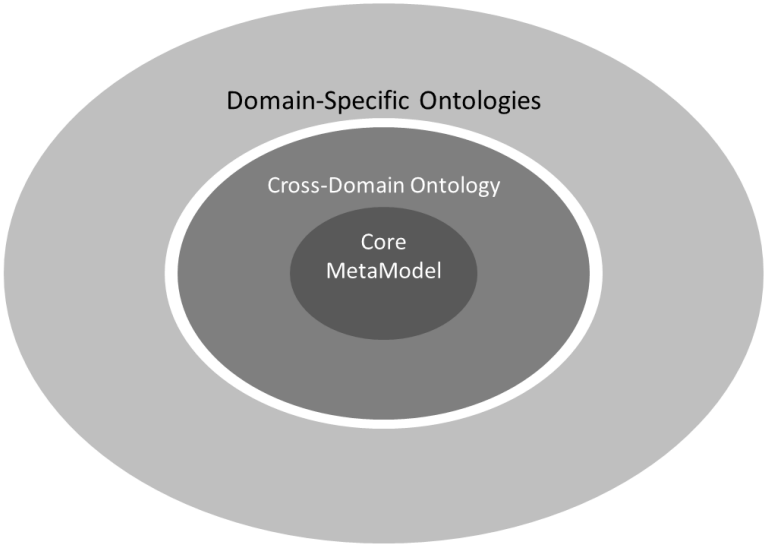

Figure 4.2.1-1: Overview of the NGSI-LD Information Model Structure

4.2.2 NGSI-LD Meta Model

Figure 4.2.2-1 provides a graphical representation of the NGSI-LD Meta-Model in terms of classes and their relationships. To provide additional clarity an informal (non-normative) mapping to the Property Graph Model is also presented.

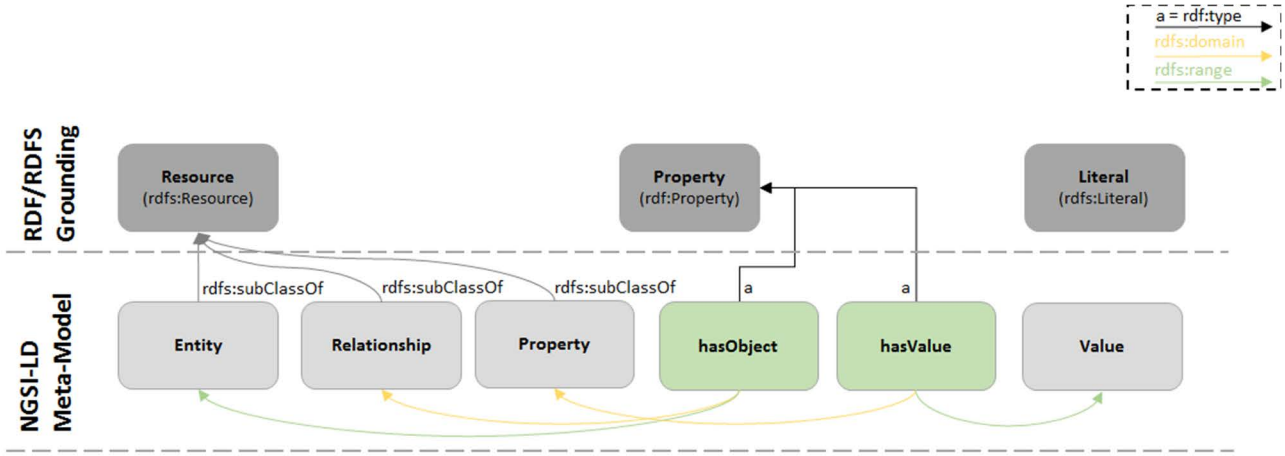

Figure 4.2.2-1: NGSI-LD Core Meta-Model

Implementations shall support the NGSI-LD Meta-model as follows:

- An **NGSI-LD Entity** is a subclass of rdfs:Resource [1].
- An **NGSI-LD Relationship** is a subclass of rdfs:Resource [1].
- An **NGSI-LD Property** is a subclass of rdfs:Resource [1].
- An **NGSI-LD Value** shall be either a rdfs:Literal or a node object (in JSON-LD language) to represent complex data structures [1].
- An **NGSI-LD Property** shall have a **value,** stated through *hasValue*, which is of type rdf:Property [1].
- An **NGSI-LD Relationship** shall have an **object** stated through *hasObject* which is of type rdf:Property [1].

4.2.3 Cross Domain Ontology

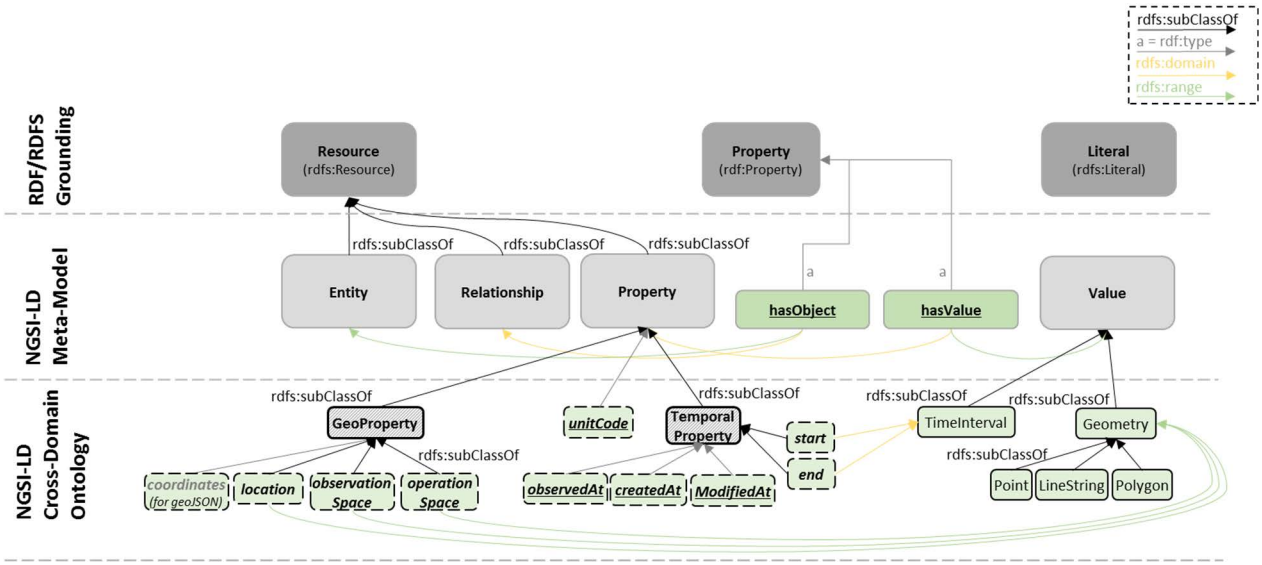

Figure 4.2.3-1: NGSI-LD Core Meta-Model plus the Cross-Domain Ontology

Figure 4.2.3-1 describes the concepts introduced by the NGSI-LD Cross-Domain Ontology, which shall be supported by implementations as follows:

- **Geo Properties:** Are intended to convey geospatial information and implementations shall support them as defined in clause 4.7.

- **Temporal Properties:** They are non-reified Properties (represented only by its Value) that convey temporal information for capturing the time series evolution of other Properties; implementations shall support them as defined in clause 4.8.
- **"unitCode" Property:** A Property intended to provide the units of measurement of an NGSI-LD Value. Implementations shall support it as defined in clause 4.5.1.
- **Geometry Values:** They are a special type of NGSI-LD Value intended to convey geometries corresponding to geospatial properties. Implementations shall support them as defined in clause 4.7.
- **Time Values:** They are a special type of NGSI-LD Value intended to convey time instants or intervals representations. Implementations shall support them as defined in clause 4.6.3.

Clause 4.4 defines the Core JSON-LD @context which includes the URIs which correspond to the concepts introduced above.

4.2.4 NGSI-LD domain-specific models and instantiation

This clause is informative and is intended to illustrate the relationship between the NGSI-LD Information Model and NGSI-LD Domain-specific models.

Figure 4.2.4-1 shows an example of an NGSI-LD domain-specific model. Domain-specific models introduce the specific entity types required for a particular domain. Figure 4.2.4-1 shows the types *Car*, *Parking*, *Street*, *Gate*. Entity types can have further subtypes, e.g. *OffStreetParking* as subtype of *Parking*.

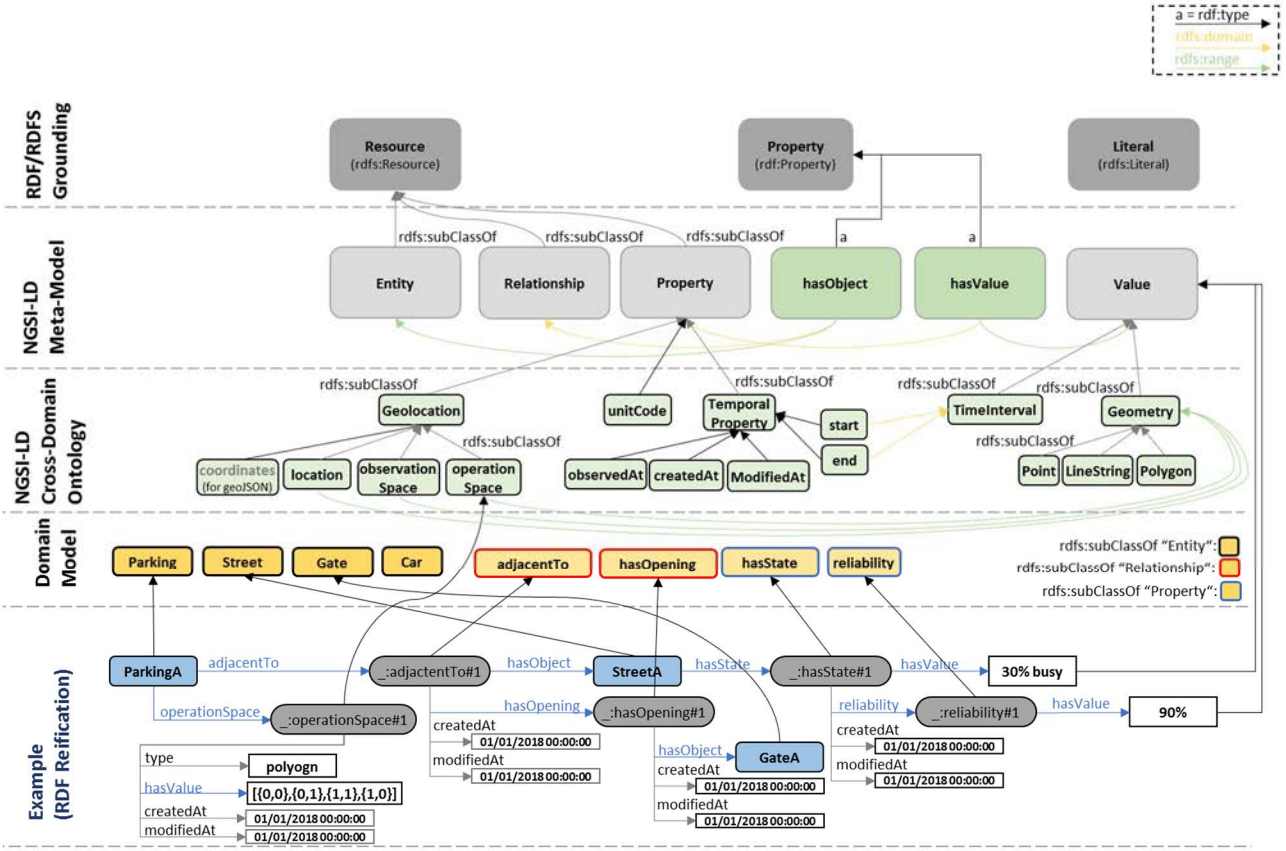

Figure 4.2.4-1: Cross-Domain Ontology and instantiation

In addition, two different NGSI-LD Properties are introduced ('*hasState*', *'reliability'*).

The *'adjacentTo'* Relationship links entities of type *'Parking'* with entities of type *'Street'*.

4.2.5 UML representation

This clause is informative and is intended to show how the NGSI-LD information model could be described using UML diagrams. The aim of this diagram is to help those readers less familiar with ontology representations or RDF [1] to understand the NGSI-LD Information Model.

In figure 4.2.5-1 NGSI-LD Entity, Relationship, Property and Value are represented as UML classes. UML associations are used to interrelate these classes while keeping the structure and semantics defined by the NGSI-LD Information Model.

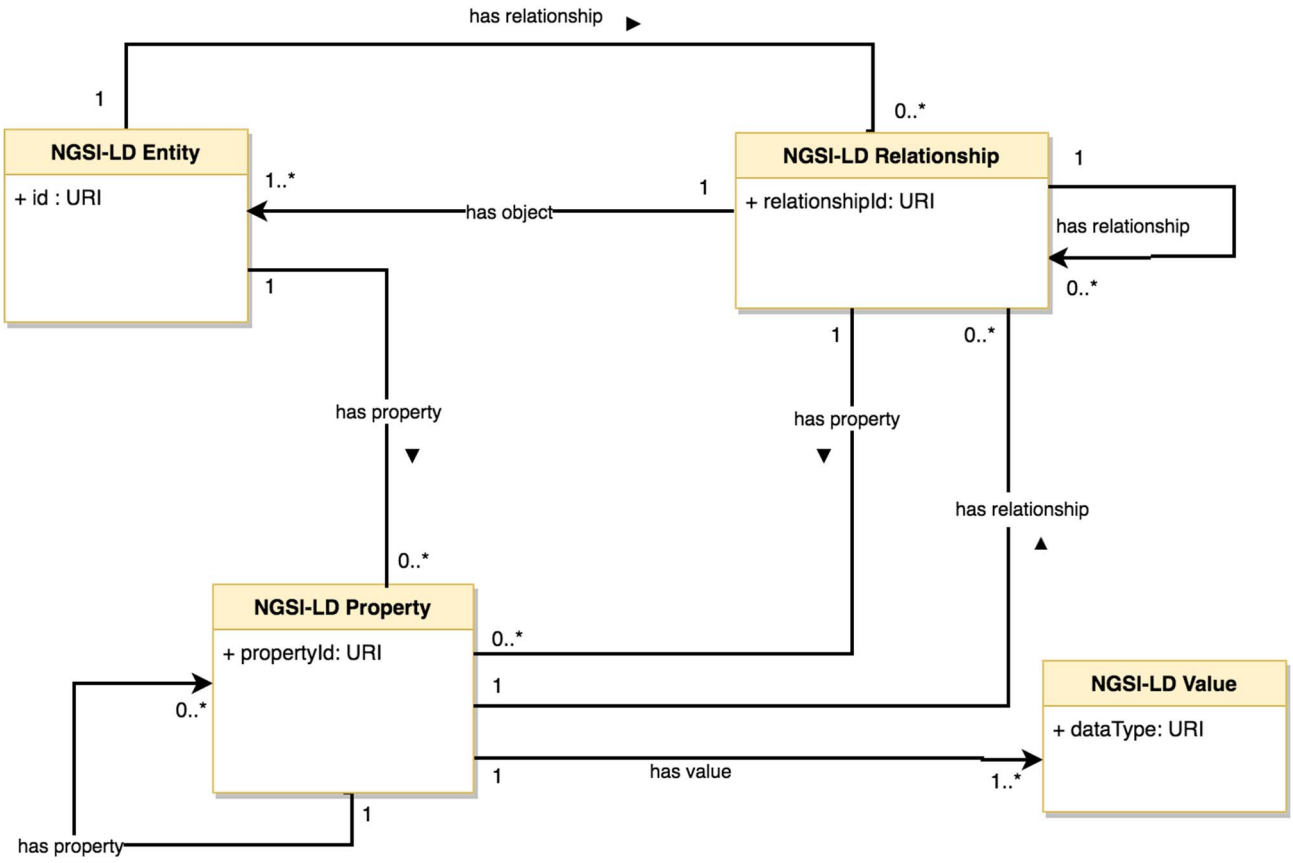

Figure 4.2.5-1: NGSI-LD information model as UML

4.3 NGSI-LD Architectural considerations

4.3.1 Introduction

The NGSI-LD API is intended to be primarily an API and does not define a specific architecture. It is envisioned that the NGSI-LD API can be used in different architectural settings and the architectural assumptions of the API are kept to a minimum.

As it is not possible to elaborate all possible architectures in which the NGSI-LD API could be used, three prototypical architectures are presented. The NGSI-LD API shall enable efficient support for all of them, i.e. the design decisions for the NGSI-LD API take these prototypical architectures into consideration. A real system architecture utilizing the NGSI-LD API can map to one, take elements from multiple or combine all of the prototypical architectures.

4.3.2 Centralized architecture

Figure 4.3.2-1 shows a centralized architecture. In the centre is a *Central Broker* that stores all the context information. There are *Context Producers* that use update operations to update the context information in the *Central Broker* and there are *Context Consumers* that request context information from the *Central Broker*, either using synchronous one-time query or asynchronous subscribe/notify operations. The *Central Broker* answers all requests from its storage. Figure 4.3.2-1 shows one component that acts as both *Context Producer* and *Context Consumer*. The general assumption is that components can have multiple roles, so such components are not explicitly shown in clauses 4.3.3 and clause 4.3.4.

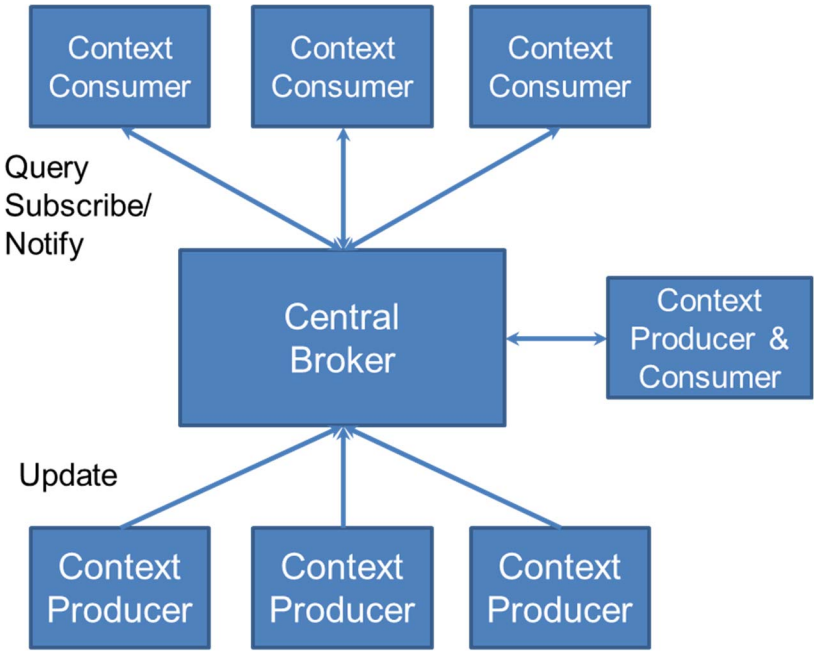

Figure 4.3.2-1: Centralized architecture

4.3.3 Distributed architecture

Figure 4.3.3-1 shows a distributed architecture. The underlying idea here is that all information is stored by the *Context Sources*. *Context Sources* implement the query and subscription part of the NGSI-LD API as a *Context Broker* does. They register themselves with the *Context Registry*, providing information about what context information they can provide, but not the context information itself, e.g. a certain *Context Source* registers that it can provide the indoor temperature for Building A and Building B or that it can provide the speed of cars in a geographic region covering the centre of a city.

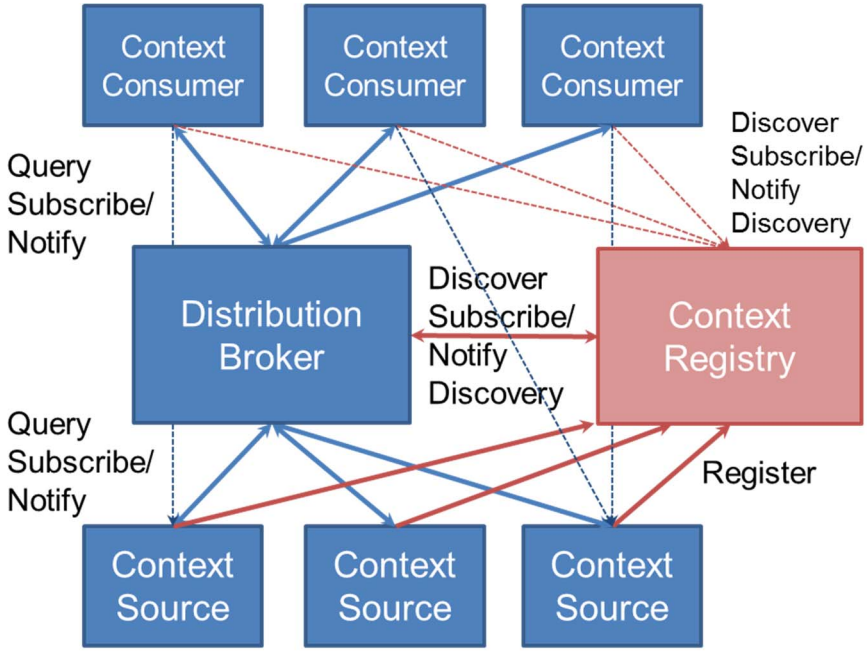

Figure 4.3.3-1: Distributed architecture

Context Consumers can query or subscribe to the *Distribution Broker*. On each request, the *Distribution Broker* discovers or does a discovery subscription to the *Registry* for relevant *Context Sources*, i.e. those that may provide context information relevant to the respective request from the *Context Consumer*. The *Distribution Broker* then queries or subscribes to each relevant *Context Source*, if possible it aggregates the context information retrieved from the *Context Sources* and provides them to the *Context Consumer*. In this mode of operation, it is not visible to the *Context Consumer*, whether the *Broker* is a *Central Broker* or a *Distribution Broker*. Alternatively, the architecture allows that *Context Consumers* can discover *Context Sources* through the *Registry* themselves and then directly request from *Context Sources*. This is shown in figure 4.3.3-1 with the fine dashed arrows.

4.3.4 Federated architecture

The federated architecture shown in figure 4.3.4-1 is used in cases where existing domains are to be federated. For example, different departments in a city operate their own Context Broker-based NGSI-LD infrastructure, but applications should be able to easily access all available information using just one point of access. The architecture works in the same way as the distributed architecture described in clause 4.3.3, except that instead of simple *Context Sources*, whole domains are registered with the respective *Context Broker* as point of access. Typically, the domains will be registered to the federation *Context Registry* on a more coarse-grained level, providing scopes, in particular geographic scopes, that can then be matched to the scopes provided in the requests. For example, instead of registering individual entities like buildings, the domain would be registered with having information about entities of type building within a geographic area. Applications then query or subscribe for entities within a geographic scope, e.g. buildings in a certain area of the city. The *Federation Broker* discovers the domain *Context Brokers* that can provide relevant information, forwards the request to these *Brokers* and aggregates the results, so the application gets the result in the same way as in the centralized and distributed cases.

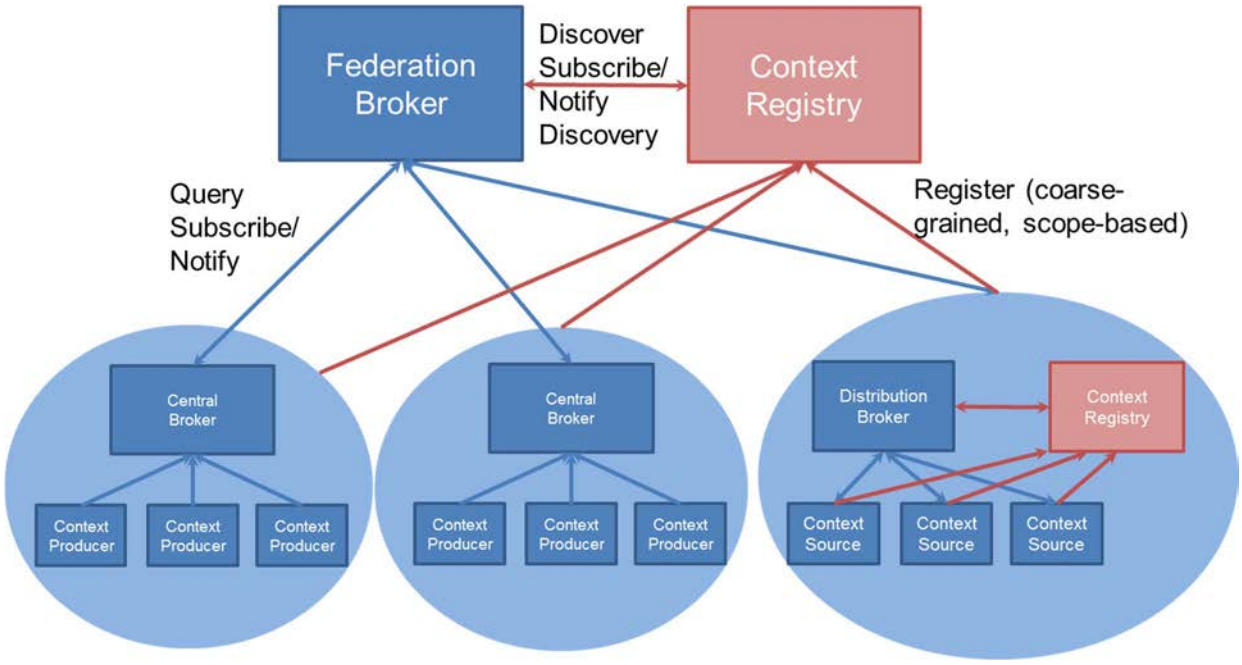

Figure 4.3.4-1: Federated architecture

A domain itself can use a centralized or distributed architecture, or could even utilize a federated architecture that federates sub-domains.

As in the distributed case, it is also possible that applications discover relevant domains through the federation-level *Context Registry* and directly contact the *Context Brokers* in the individual domains.

4.4 Core NGSI-LD @context

NGSI-LD serialization is based on JSON-LD [2], a JSON-based format to serialize Linked Data. The @context in JSON-LD is used to expand terms, provided as short hand strings, to concepts, specified as URIs, and vice versa, to compact URIs into terms. The Core NGSI-LD (JSON-LD) @context is defined as a JSON-LD @context which contains:

- The core terms needed to uniquely represent the key concepts defined by the NGSI-LD Information Model, as mandated by clause 4.2.
- The terms needed to uniquely represent all the members that define the API-related Data Types, as mandated by clause 5.2 and clause 5.3.
- A fallback *@vocab* rule to expand or compact user-defined terms to a default URI, in case there is no other possible expansion or compaction as per the current @context.

NGSI-LD compliant implementations shall support such Core @context, which shall be implicitly present when processing or generating context information. Furthermore, the Core @context is protected and shall remain immutable and invariant during expansion or compaction of terms. Therefore, and as per the JSON-LD processing rules [2], when processing NGSI-LD content, implementations shall consider the Core @context as if it were in the **last** position of the @context array. Nonetheless, for the sake of compatibility and cleanness, data providers should generate JSON-LD content that conveys the Core @context in the last position.

For the avoidance of doubt, when rendering NGSI-LD Elements, the Core @context **shall always be treated** as if it had been originally placed **in the last position**, so that, if needed, upstream JSON-LD processors can properly expand as NGSI-LD or override the resulting JSON-LD documents provided by API implementations.

The NGSI-LD Core @context is publicly available at https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld and shall contain all the terms as mandated by annex B.

4.5 NGSI-LD Data Representation

4.5.1 NGSI-LD Entity Representation

An NGSI-LD Entity shall be represented by an object encoded using JSON-LD [2]. The rules described below state the encoding that shall be supported by implementations. Annex D provides a computational description of this process in terms of an algorithm.

In addition to the terms defined by the Core NGSI-LD @context (mandatory as per annex B), the @context should contain the following terms:

- One term associated to the Entity Type, mapping the Entity Type Name with its Type Identifier (URI).
- One term associated to the name of each Property used by the entity representation (see below), mapping the Property Name with its Property Identifier (URI). If the Property's range is a data type different than a native JSON type, then it shall be conveyed explicitly under this term by using a nested JSON object in the form:
  - "@type": <Datatype's URI>.
  - "@id": <Property's URI>.

NOTE 1: To support the case of multiple instances of a Property with the same Property Identifier, multiple elements with the same name plus an index can appear. For details, see clause 4.5.5.

EXAMPLE: Mapping indexed Property names to the same Property Identifier:

- "speed#1": "http://example.org/speed";
- "speed#2": "http://example.org/speed".

- One term associated to the name of each Relationship used by the entity representation, mapping the Relationship Name with the Relationship Identifier (URI) in the form:
  - "@type": "@id".
  - "@id": <Relationship's URI>.

NOTE 2: To support the case of multiple instances of a Relationship with the same Relationship Identifier, multiple elements with the same name plus an index can appear. For details, see clause 4.5.5.

The JSON-LD object shall contain at least the following members:

- "id" whose value shall be a URI that identifies the Entity.
- "type" whose value shall be equal to the Entity Type Name.
- "@context" as mandated by [2], section 5.1. Depending on the binding, the @context may not just be provided in line with rest of the JSON content, but there could be other options. For example, in the HTTP binding, the @context can be made available through a Link header (see clause 6.3.5).
- One or more (in case of indexed Properties) members for each Property as per the rules stated in clause 4.5.2.
- One or more (in case of indexed Relationships) members for each Relationship as per the rules stated in clause 4.5.3.

4.5.2 NGSI-LD Property Representation

An NGSI-LD Property shall be represented by a member whose key is the Property Name (a term), possibly with an index, and whose value is a JSON-LD object including the following members:

- "type": "Property". *Mandatory*.
- "value": the Property Value (see definition in clause 3.1). *Mandatory*. If the Value's datatype is a native JSON data type it shall be encoded directly as the member's value. Otherwise the member's value shall be a JSON object in the form:
  - "@type": <Data Type URI>.
  - "@value": Property Value.
- "observedAt": a string as mandated by clause 4.8. *Optional*.
- "datasetId": a URI as mandated by clause 4.5.5. *Optional.*
- "instanceId": a URI uniquely identifying a Property instance. *System generated. Optional.*

NOTE: For temporal representations, systems should maintain an *instanceId* for each Property instance. Without such an *instanceId*, it is not possible to selectively modify or delete temporal information via the NGSI-LD API. The consequences of this may be severe in the case of modification or deletion requests for legal reasons, e.g. GDPR [i.18]. When implementing the NGSI-LD API on storage systems that do NOT allow modification or deletion, similar problems may be encountered.

- "createdAt": a string as mandated by clause 4.8. *System generated.*
- "modifiedAt": a string as mandated by clause 4.8. *System generated.*
- "unitCode": a string representing the measurement unit corresponding to the Property value. It shall be encoded using the UN/CEFACT Common Codes for Units of Measurement [15]. *Optional*.
- For each of the Properties this Property is associated with, a member whose key (a term) is the Property Name and value is the result of serializing a **Property**.
- For each of the Relationships this Property is associated with, a member whose key (a term) is the Relationship Name and value is the result of serializing a **Relationship**.

4.5.3 NGSI-LD Relationship Representation

An NGSI-LD Relationship shall be represented by a member whose key is the Relationship Name (a term), possibly with an index, and whose value is a JSON-LD object with the following terms:

- "type": "Relationship". *Mandatory.*
- "object": the Relationship's object represented by a URI. *Mandatory*.
- "observedAt": a string as mandated by clause 4.8. *Optional*.
- "datasetId": a URI as mandated by clause 4.5.5. *Optional.*
- "instanceId": a URI uniquely identifying a Relationship instance. *System generated. Optional.*

NOTE: For temporal representations, systems should maintain an *instanceId* for each Property instance. Without such an *instanceId*, it is not possible to selectively modify or delete temporal information via the NGSI-LD API. The consequences of this may be severe in the case of modification or deletion requests for legal reasons, e.g. GDPR [i.18]. When implementing the NGSI-LD API on storage systems that do NOT allow modification or deletion, similar problems may be encountered.

- "createdAt": a string as mandated by clause 4.8. *System generated*.
- "modifiedAt": a string as mandated by clause 4.8. *System generated*.

- For each of the Relationships this Relationship is associated with, a member whose key is the Relationship Name (a term) and whose value is the result of serializing a Relationship as per the rules of representation of a **Relationship**.
- For each of the Properties this Relationship is associated with, a member whose key is the Property Name (a term) and whose value is the result of serializing a Property as per the rules of representation of a **Property**.

4.5.4 Simplified Representation

The NGSI-LD specification defines an alternative, abbreviated representation of Entities, which allows consuming only entity data (the target object of each Relationship or the value of each Property) corresponding to the Properties or Relationships whose subject is the Entity itself i.e. the own Attributes of the Entity. The simplified representation of Entities shall be supported by implementations and can be selected by Context Consumers through specific request parameters. An example of this representation can be found in annex C, clause C.2.2.

The simplified representation of an entity shall include the following:

- A JSON-LD @context as described in clause 4.4.
- A JSON-LD object containing the following members:
  - Id, type and @context as described in clause 4.4.
  - For each **Property** a member whose key is the Property Name (a term) and whose value is the Property Value.
  - For each **Relationship** a term whose key is the Relationship Name (a term) and whose value is the Relationship's Object (represented as a URI).

4.5.5 Multi-Attribute Support

For each Entity, there can be Properties and Relationships that simultaneously have more than one instance. In the case of Properties, there may be more than one source at a time that provides a Property instance, e.g. based on independent sensor measurements with different quality characteristics. For instance, take a speedometer and a GPS both providing the current speed of a car. In the case of Relationships, there may be non-functional Relationships, e.g. for a room, there may be multiple "contains" Relationships to all sorts of objects currently in the room that have been put there by different people and which are dynamically changing over time.

To be able to explicitly manage such multi-attributes, the optional *datasetId* property is used, which is of datatype URI. It is introduced for Properties and Relationships in clauses 4.5.2 and 4.5.3 respectively. If a *datasetId* is provided when creating, updating, appending or deleting Properties and Relationships, only instances with the same *datasetId* are affected, leaving instances with another *datasetId* or an instance without a *datasetId* untouched. An example can be found in annex C, clause C.2.2.

When requesting Entity information, multiple instances of fitting Properties and Relationships are returned as different JSON members, i.e. using Property and Relationship Names with an additional index (as specified in clause 4.6.2), but they all map to the same Property or Relationship URI. This mapping is specified in the respective JSON-LD @context.

4.5.6 Temporal Representation of an Entity

The temporal representation of an Entity shall be as mandated by clause 4.5.1, but for each Property and Relationship their temporal representation shall be provided as mandated by clauses 4.5.7 and 4.5.8.

4.5.7 Temporal Representation of a Property

The temporal representation of an NGSI-LD Property (for instance, its historical evolution) is composed of the sequence of instances of the referred Property during a period of time within its lifetime.

The temporal evolution of an NGSI-LD Property shall be represented as an Array of JSON-LD objects, each one representing an instance of the Property (as mandated by clause 4.5.2) at a particular point in time, which is recorded as a Temporal Property of the instance (typically "observedAt"). See example in annex C, clause C.5.6.

If a Property is static (i.e. it has not changed over time) then it shall be represented by an Array with a single instance.

4.5.8 Temporal Representation of a Relationship

The temporal representation of an NGSI-LD Relationship (for instance, its historical evolution) is composed of the sequence of instances of the referred Relationship during a period of time within its lifetime.

The temporal evolution of an NGSI-LD Relationship shall be represented as an Array of JSON-LD objects, each one representing an instance of the Relationship (as mandated by clause 4.5.3) at a particular point in time, which is recorded as a Temporal Property of the instance (typically "observedAt"). See example in annex C, clause C.5.5.

If a Relationship is static (i.e. it has not changed over time) then it shall be represented by an Array with a single instance.

4.5.9 Simplified Temporal Representation of an Entity

The NGSI-LD specification defines an alternative, abbreviated temporal representation of Entities, which allows consuming temporal Entity data in a more straightforward manner. The simplified temporal representation of Entities shall be supported by implementations and can be selected by Context Consumers through specific request parameters. An example can be found in annex C, clause C.5.6.

The simplified temporal representation of an Entity shall include the following:

- A JSON-LD @context as described in clause 4.4.
- A JSON-LD object containing the following members:
  - *id*, *type* and *@context* as described in clause 4.4.
  - For each **Property** a member whose key is the Property Name (a term). The member value shall be a JSON-LD object labelled with the type "Property". Such JSON-LD object shall only contain a member whose key shall be "values". The value of the referred *values* member shall be a JSON-LD Array that shall contain as many array elements as Property instances (i.e. data points of the concerned Property) being represented. Each array element shall be another Array containing exactly two array elements: the first element shall be a Property value and the second element shall correspond to the associated Temporal Property (for instance "observedAt"). If the value of the Temporal Property is not known then an empty string ("") shall be conveyed as the second array element.
  - For each **Relationship** a term whose key is the Relationship Name (a term). The member value shall be a JSON-LD object labelled with the type "Relationship". Such JSON-LD object shall only contain a member whose key shall be "objects". The value of the referred *objects* member shall be a JSON-LD Array that shall contain as many array elements as Relationship instances (i.e. data points of the concerned Relationship) being represented. Each array element shall be another array containing exactly two elements: the first element shall be a Relationship object (a URI) and the second element shall correspond to the associated Temporal Property (for instance "observedAt"). If the value of the Temporal Property is not known then an empty string ("") shall be conveyed as the second array element.

4.6 Data Representation Restrictions

4.6.1 Supported text encodings

NGSI-LD implementations shall support the **UTF-8** text encoding format. To avoid interoperability problems, applications shall provide JSON content encoded using UTF-8 and NGSI-LD systems shall also expose such JSON content using UTF-8.

4.6.2 Supported names

Even though the JSON serialization format allows inclusion of any character in the Unicode space, for the sake of maximizing interoperability, NGSI-LD restricts Entity Type Names, Property Names and Relationship Names to the following ABNF grammar:

- nameChar =/ DIGIT / ALPHA
- nameChar =/ %x5F ; _
- name = 1*nameChar

In order to support multi-attributes as specified in clause 4.5.5, an optional index can be added to Property Names and Relationship Names based on the following ABNF grammar:

- name =/ 1*nameChar %x23 1*DIGIT; name#1*DIGIT

In order to avoid name clashing, names can be prefixed as specified by the following BNF grammar:

- prefix = 1*nameChar
- name =/ prefix %x3A 1*nameChar; prefix:name

When receiving a JSON-LD object with a Name (Type, Property, Relationship) including characters different than those expressed above, implementations shall raise an error of type *BadRequestData*.

4.6.3 Supported data types for Values

Compliant NGSI-LD implementations shall support the following data types for representing Values:

- All the JSON native data types as mandated by IETF RFC 8259 [6], section 3.
- All the GeoJSON *Geometries* [8] with the exception of *GeometryCollection*.
- **DateTime** string for encoding a timestamp, i.e. a calendar date together with a time of day, expressed in **UTC**, using the ISO 8601 [17] Complete Representation and in particular using the 'Extended Format', as described below:
  - The timestamp shall be a string containing *Year*, *Month*, *Day*, *Hours*, *Minutes*, *Seconds and time zone* components using the format *YYYY-MM-DDThh:mm:ssZ* as defined in ISO 8601 [17]. In this representation, the character "-" is used to separate the calendar date components, the character "T" is used to indicate the start of the time of day portion, the character ":" is used to separate the time of day components, and the trailing character "Z" is used to convey the time zone.
  - All the referred components shall appear in the string; reduced representations are not permitted.
  - The *Seconds* component may optionally contain a decimal fraction. In this case the string shall contain two integer digits, followed by a comma and then one or more fractional digits, up to a maximum of six. For example, *YYYY-MM-DDThh:mm:ss,ssssssZ.*
  - The trailing timestamp component shall contain the time zone related information and shall always be equal to the character "Z". Therefore, all timestamps shall be expressed in **UTC**.
- **Date** string for encoding a calendar date. It uses ISO 8601 [17] Complete Representation using the 'Extended Format', as described below:
  - It shall be a string containing *Year*, *Month*, *Day* components using the format *YYYY-MM-DD* as defined in ISO 8601 [17]. In this representation, the character "-" is used to separate the calendar date components.
  - All the referred components shall appear in the string; reduced representations are not permitted.

- **Time** string for encoding a local time expressed in **UTC**. It uses ISO 8601 [17] Complete Representation using the 'Extended Format', as described below:
  - It shall be a string containing *Hours*, *Minutes* and *Seconds* components using the format *hh:mm:ssZ* as defined in ISO 8601 [17]. In this representation, the character ":" is used to separate the local time components.
  - All the referred components shall appear in the string; reduced representations are not permitted.
  - The *Seconds* component may optionally contain a decimal fraction. In this case the string shall contain two integer digits, followed by a comma and then one or more fractional digits, up to a maximum of six. For example, *hh:mm:ss,ssssssZ.*
  - The string shall not contain expressions of the difference between local time and UTC. All representations shall be interpreted as being expressed in **UTC**.
- URI as mandated by ISO 8601 [17], Appendix A, production rule named 'URI'.

Implementations may support additional data types different to those enumerated above, for instance:

- JSON-LD typed value (i.e. a string as the lexical form of the value together with a type, defined by an XSD base type or more generally an IRI).
- JSON-LD structured value (i.e. a set, a list, a language-tagged string).

4.6.4 Supported Entity Content

In principle, context information providers can publish any kind of data serialized in JSON and encoded in UTF-8. Nonetheless, to avoid security problems caused by script injection attacks or other attack vectors, the following characters are **prohibited** and shall not be part of any value:

- %x3C ; <
- %x3E ; >
- %x22 ; "
- %x27 ; '
- %x3D ; =
- %x3B ; ;
- %x28 ; (
- %x29 ;)

When receiving entities (context information) encoded in JSON format and containing values that include the forbidden characters implementations shall raise an error of type *BadRequestData*.

4.7 Geospatial Properties

4.7.1 GeoJSON Geometries

Geospatial Properties in NGSI-LD shall be represented using **GeoJSON** Geometries [8]. With the aim of highlighting and encoding those Properties which convey geospatial characteristics, NGSI-LD defines a special type of Property named *GeoProperty*, defined by the NGSI-LD @context described by the present document in clause 4.4.

When dealing with NGSI-LD Entities, implementations shall interpret JSON-LD nodes of type *GeoProperty* just as conventional Properties but with the additional requirement that the Value corresponding to such Property shall be a GeoJSON Geometry. All the Geometries defined by [8] are allowed except *GeometryCollection*. In addition, implementations should take the necessary steps to create the corresponding geo-indexes so that information can be properly returned when geo-queries are executed.

NGSI-LD defines the following Properties of type *GeoProperty*. Preferably these Properties should be used if they semantically fit, but if necessary, additional Properties of type *GeoProperty* can be defined by Context Producers:

- **location** is defined as the geospatial Property representing the geographic location of the Entity, e.g. the location of a building or the current location of a car.
- **observationSpace** is defined as the geospatial Property representing the geographic location that is being observed, e.g. by a sensor. For example, in the case of a camera, the location of the camera and the observation space are different and can be disjoint.
- **operationSpace** is defined as the geospatial Property representing the geographic location in which an Entity, e.g. an actuator is active. For example, a crane can have a certain operation space.

The defined Properties can also be used as part of Context Source Registrations (see clause 5.2.9). In this case they represent locations in which Entities with the respective geospatial Properties are contained. For example, a Context Source that monitors the location of cars in a city may be represented by a Context Source Registration whose Property *location* corresponds to the space of the city in which the location of cars is monitored.

4.7.2 Representation of GeoJSON Geometries in JSON-LD

There are certain types of GeoJSON geometries, for instance *Polygon*, whose coordinates are represented using nested array structures (through the *coordinates* member). Such representation may introduce serialization problems when transforming JSON-LD content into RDF graphs.

Also, when using whole GeoJSON geometries (consisting of *type* and *coordinates*) in an NGSI-LD document, its JSON syntax is only preserved in the regular JSON-LD representation (with separate @context), but not in an expanded representation. To handle resulting problems, optionally, whole GeoJSON geometries can be represented as a JSON string.

Implementations shall accept the referred encoded string value, if and only if, it can be parsed into a JSON Object, as mandated by IETF RFC 8259 [6], meeting the syntax and restrictions mandated by IETF RFC 7946 [8] when representing a valid Geometry of the type specified.

For the avoidance of doubt, regular encodings of GeoJSON geometries (as JSON Object) shall also be accepted by implementations, but Context Producers should consider the implications in terms of RDF compatibility.

4.8 Temporal properties

NGSI-LD defines the following Properties of type *TemporalProperty* that shall be supported by implementations:

- **observedAt** is defined as the temporal Property at which a certain Property or Relationship became valid or was observed. For example, a temperature Value was measured by the sensor at this point in time.
- **createdAt** is defined as the temporal Property at which the Entity, Property or Relationship was entered into an NGSI-LD system.
- **modifiedAt** is defined as the temporal Property at which the Entity, Property or Relationship was last modified in an NGSI-LD system, e.g. in order to correct a previously entered incorrect value.

Temporal Properties in NGSI-LD shall be represented based on the *DateTime* data type as mandated by clause 4.6.3.

NOTE 1: For simplicity reasons, a *TemporalProperty* is represented only by its Value, i.e. no Properties of *TemporalProperty* nor Relationships of *TemporalProperty* can be conveyed. In more formal language, a *TemporalProperty* does not allow reification.

NOTE 2: It is important to remark that the term *TemporalProperty* has been reserved for the semantic tagging of non-reified structural timestamps (*observedAt*, *createdAt*, *modifiedAt*), which capture the temporal evolution of Entity Attributes. Only such structural timestamps can be used as *timeproperty* in Temporal Queries as mandated by clause 4.11.

NOTE 3: User-defined Properties whose value is a time value (*Date, DateTime or Time*) are defined as *Property*, not as *TemporalProperty*, and are serialized in NGSI-LD as shown in clause C.6.

4.9 NGSI-LD Query Language

The NGSI-LD Query Language shall be supported by implementations. It is intended to:

- filter out Entities by Attribute Values (target value of a Property or the target object of a Relationship);
- filter out Context Sources by the values of properties that describe them (defined when Context Sources are registered).

The grammar that defines the query language in ABNF format [12] is described below (it has been validated using https://tools.ietf.org/tools/bap/abnf.cgi) and shall be supported by implementations:

```
Query = (QueryTerm / QueryTermAssoc) *(logicalOp (QueryTerm / QueryTermAssoc))
QueryTermAssoc = %x28 QueryTerm *(logicalOp QueryTerm) %x29          ; (QueryTerm)
QueryTerm = Attribute
QueryTerm = Attribute Operator ComparableValue
QueryTerm =/ Attribute equal CompEqualityValue
QueryTerm =/ Attribute unequal CompEqualityValue
QueryTerm =/ Attribute patternOp RegExp
QueryTerm =/ Attribute notPatternOp RegExp
Attribute = attrName / compoundAttrName / attrPathName
Operator = equal / unequal / greaterEq / greater / lessEq / less
ComparableValue = Number / quotedStr / dateTime / date / time
OtherValue = false / true
Value = ComparableValue / OtherValue
Range = ComparableValue dots ComparableValue
ValueList = Value 1*(%x2C Value)                                     ; Value 1*(, Value)
CompEqualityValue = OtherValue / ValueList / Range / URI
equal = %x3D %x3D                                                    ; ==
unequal = %x21 %x3D                                                  ; !=
greater = %x3E                                                       ; >
greaterEq = %x3E %x3D                                                ; >=
less = %x3C                                                          ; <
lessEq = %x3C %x3D                                                   ; <=
patternOp = %x7E %x3D                                                ; ~=
notPatternOp = %x21 %x7E %x3D                                        ; !~=
dots = %x2E %x2E                                                     ; ..
attrNameChar =/ DIGIT / ALPHA
attrNameChar =/ %x5F                                                 ; _
attrName = 1*attrNameChar
attrPathName = attrName *(%x2E attrName)                             ; attrName *(. attrName)
compoundAttrName = attrName *(%x5B (attrName) %x5D)                  ; . attrName *([ attrName ])
quotedStr = String                                                   ; '*char'
andOp = %x3B                                                         ; ;
orOp = %x7C                                                          ; |
logicalOp = andOp / orOp
```

- *DIGIT* and *ALPHA* are defined by IETF RFC 5234 [12].
- *Number* shall be a number as mandated by the JSON Specification, following the ABNF Grammar, production rule named *number*, section 6 of IETF RFC 8259 [6].
- *String* shall be a text string as mandated by the JSON Specification, following the ABNF Grammar, production rule named *String*, section 7 of IETF RFC 8259 [6].
- *char* shall be a character as mandated by the JSON Specification, ABNF Grammar, production rule named *char*, section 7 of IETF RFC 8259 [6].
- *false* shall be conformant with the JSON ABNF Grammar, production rule named *false*, section 3 of IETF RFC 8259 [6]. It is intended to represent the Boolean value corresponding to "false".
- *true* shall be conformant with the JSON ABNF Grammar, production rule named *true*, section 3 of IETF RFC 8259 [6]. It is intended to represent the Boolean value corresponding to "true".
- *RegExp* shall be a regular expression as mandated by IEEE POSIX 1003.2™ [11].
- *dateTime* shall be a *DateTime* value as mandated by clause 4.6.3.
- *time* shall be a *Time* value as mandated by clause 4.6.3.
- *date* shall be a *Date* value as mandated by clause 4.6.3.

- *URI* shall be a URI as mandated by IETF RFC 3986 [5], appendix A, production rule named *URI*.

A **Query Term** (production rule *QueryTerm*) defines a predicate which serves as a matching condition for Entities. The constituent parts of a Query Term are:

- an attribute path (production rule named *Attribute*);
- an optional pair composed by an operator (production rule named *Operator*) and a value (production rule named *Value*).

EXAMPLE 1: temperature==20.

EXAMPLE 2: temperature.observedAt>=2017-12-24T12:00:00Z.

EXAMPLE 3: brandName!= "Mercedes".

EXAMPLE 4: isParked=="urn:ngsi-ld:OffStreetParking:Downtown1".

EXAMPLE 5: A query encoded as an HTTP Query String. Please note that this is HTTP binding specific.

?type=Vehicle&q=speed>50;brandName!=Mercedes.

EXAMPLE 6: isMonitoredBy (to query Entities that have the Attribute *isMonitoredBy*).

Query Terms may be combined through logical operators that shall be supported by implementations as follows:

- The production rule *andOp* defines a logical AND operator conveying that the requested entities are those which meet at the same time the conditions posed by all the Query Terms affected by such an operator.
- The production rule *orOp* defines a logical OR operator conveying that the requested entities are those which meet any of the conditions posed by the Query Terms affected by such an operator.
- When evaluating logical conditions, and in the absence of specific Query Term associations (see below), the logical AND operator shall take precedence over the logical OR operator.

Association of Query Terms shall be supported by implementations as per the grammar included by the present clause (production rule named *QueryTermAssoc*). An association of Query Terms is composed of the combination of different Query Terms linked by logical operators (AND, OR) and delimited by parenthesis. The evaluation of an association of Query Terms shall always take precedence over individual, non-associated Query Terms.

EXAMPLE 7: ((speed>50|rpm>3000);brandName=="Mercedes").

EXAMPLE 8: (temperature>=20;temperature<=25)|capacity<=10.

The syntax of an attribute path is defined by the production rule *Attribute*, as a list of names. Such a list is intended to address a Property or Relationship included by the matching entities subjacent graph, in accordance with the following rules:

- Every name in the list shall be expanded to a URI (fully qualified name) as mandated by clause 5.5.7.
- The first name shall refer to a Property or Relationship (top level element) whose subject shall be a matching Entity. Strictly speaking, and as per the JSON-LD representation rules, such (fully qualified) name shall be equal to the (fully qualified) name of the concerned Property or Relationship.
- Each other name (if present) represents a (sub)Property or (sub)Relationship, starting with the top-level element as subject and continuing through the graph traversal. The element addressed by the last name in the list is defined as the target element. If only one name is present in the attribute path, then the target element is the top level one.

If the target element is a Property, the **target value** is defined as the Value associated to such Property. If a Property has multiple instances (identified by its respective *datasetId*), and no *datasetId* is explicitly addressed, the target value shall be any Value of such instances.

If the target element is a Relationship, the **target object** is defined as the object associated (represented as a URI) to such Relationship. If a Relationship has multiple instances (identified by its respective *datasetId*), and no *datasetId* is explicitly addressed, the target object shall be any object of such instances.

When a Query Term only defines an attribute path (production rule named *Attribute*), the matching Entities shall be those which define the target element (Property or a Relationship), regardless of any target value or object.

Lastly, implementations shall support queries involving specific data subitems belonging to a Property Value (**seed target value**) represented by a JSON object structure (complex value). For that purpose, an attribute path may contain a **trailing path** (production rule named *compoundAttrName*) composed of a concatenated list of JSON member names, each one enclosed in between square brackets, and intended to address a specific data subitem (member) within the **seed target value.** When such a trailing path is present, implementations shall interpret and evaluate it (against the seed target value) as a *MemberExpression* of ECMA 262 [21]. If the evaluation of such *MemberExpression* does not result in a defined value, the target element shall be considered as non-existent for the purpose of query resolution.

EXAMPLE 9: address[addressLocality]== "Berlin". The trailing path is [addressLocality] and is used to refer to a particular subitem within a Postal Address.

If the target element corresponds to a Relationship, the combination of such target element with any operator different than *equal* or *unequal* shall result in **not matching**.

A **Query Term value** shall be any of the following (depending on the operator used):

- A literal value (string, number, date, etc.) (production rule named *Value*).
- A range of values (production rule named *Range*), specified as a minimum and a maximum value.
- A regular expression (production rule named *RegExp*).
- A URI (production rule named *URI*).
- A comma-separated list of literal values (production rule named *ValueList*).

When comparing dates or times, the order relation considered shall be a temporal one.

When it comes to comparing text strings, implementations:

- shall follow the recommendations defined by IETF RFC 8259 [6], section 8.3;
- should support the Unicode Collation Algorithm (UCA), as defined by [13].

URI comparison should be performed so that the number of false negatives is minimized, as recommended by IETF RFC 3986 [5], section 6.

The semantics of the different logical operators used by Query Terms are described as follows and shall be supported by compliant implementations:

- **Equal** operator (production rule named *equa*l). A matching Entity shall contain the target element and meet any of the following conditions:
  - The Query Term value, e.g. color == "red":
    - Is identical or equivalent to the target value (e.g. matches "red").
    - Is included in the target value, if the latter is an array (e.g. matches ["blue","red","green"]).
  - If the Query Term value is a list of values (production rule named *ValueList*), e.g. color=="black", "red":
    - The target value is identical or equivalent to any of the list values (e.g. matches "red").
    - The target value includes any of the Query Term values, if the target value is an array (e.g. matches ["red","blue"]).
  - If the Query Term value is a range (production rule named *Range*), e.g. temperature==10..20:
    - The target value is in the interval between the minimum and maximum of the range (both included) (e.g. matches 15).
  - If there is no equality between the target value data type and the Query Term value data type, then it shall be considered as not matching.

- **Unequal** operator (production rule named *unequal*). A matching entity shall contain the target element and meet any of the following conditions:
  - The Query Term value, e.g. color!= "red":
    - Is neither identical nor equivalent to the target value (e.g. matches "black").
    - Is not included in the target value, if the latter is an array (e.g. matches ["blue","black","green"], but not ["blue","red","green"]).
  - If the Query Term value is a list of values (production rule named *ValueList*), e.g. color!= "black", "red":
    - The target value is neither identical nor equivalent to any of the list values (e.g. matches "blue").
    - The target value does not include any of the list values, if the target value is an array (e.g. matches ["blue","yellow","green"], but not ["blue","red","green"]).
  - If the Query Term value is a range (production rule named *Range*), e.g. temperature!=10..20:
    - The target value is not in the interval between the minimum and the maximum (both included) (e.g. matches 9).
  - If the data type of the target value and the data type of the Query Term value are different, then they shall be considered unequal.
- **Greater than** operator (production rule named *greater*). For an entity to match, it shall contain the target element and the target value has to be strictly greater than the Query Term value:
  - If there is no equality between the target value data type and the Query Term value data type then it shall be considered as not matching.
- **Less than** operator (production rule named *less*). For an entity to match, it shall contain the target element and the target value shall be strictly less than the value:
  - If there is no equality between the target value data type and the Query Term value data type then it shall be considered as not matching.
- **Greater or equal than** (production rule named *greaterEq*). A matching entity shall meet any of the *Greater than* or the *Equal* conditions for single values.
- **Less or equal than** (production rule named *lessEq*). A matching entity shall meet any of the *Less than* or the *Equal* conditions for single values.
- **Match pattern** (production rule named *patternOp*). A matching entity shall contain the target element and the target value shall be in the L(R) of the regular pattern specified by the Query Term:
  - If the target value data type is different than String then it shall be considered as not matching.
- **Do not match pattern** (production rule named *notPatternOp*). A matching entity shall contain the target element and the target value shall not be in the L(R) of the regular pattern specified by the Query Term:
  - If the target value data type is different than String then it shall be considered as not matching.

4.10 NGSI-LD Geo-query language

The NGSI-LD Geo-query language shall be supported by implementations. It is intended to define predicates which allow testing whether a specific topological spatial relationship exists between a pair of geometries: a target geometry and a reference geometry. The target geometry represents a geospatial Property of an Entity, typically, the location of the Entity.

The following grammar defines the syntax for the geospatial relationships that shall be supported:

```
andOp = %x3B                        ; ;
equal = %x3D %x3D                   ; ==
georel = nearRel / withinRel / containsRel / overlapsRel / intersectsRel / equalsRel / disjointRel
nearRel = nearOp andOp distance equal PositiveNumber   ; near;max(min)Distance==x (in meters)
distance = "maxDistance" / "minDistance"
nearOp = "near"
withinRel = "within"
containsRel = "contains"
intersectsRel = "intersects"
equalsRel = "equals"
disjointRel = "disjoint"
overlapsRel = "overlaps"
```

PositiveNumber shall be a non-zero positive number as mandated by the JSON Specification. Thus, it shall follow the ABNF Grammar, production rule named *Number*, section 6 of IETF RFC 8259 [6], excluding the 'minus' symbol and excluding the number 0.

Reference geometries shall be specified by:

- A geometry type (parameter name **geometry**) as defined by the GeoJSON specification (IETF RFC 7946 [8], section 1.4), except *GeometryCollection*.
- A coordinates (parameter name **coordinates**) element which shall represent the coordinates of the reference geometry as mandated by IETF RFC 7946 [8], section 3.1.1.

The *GeoProperty* to which the geo-query is to be applied can be specified by an extra parameter named **geoproperty**. If no *geoproperty* is specified, the geo-query is applied to the default Property *location* (see clause 4.7.1).

(Please note that proper URL encoding shall be used by HTTP binding API clients when using these examples.)

EXAMPLE 1: georel=near;maxDistance==2000

geometry=Point

coordinates=[8,40]

geoproperty=observationSpace

EXAMPLE 2: georel=within

geometry=Polygon

coordinates= [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]]]

geoproperty=location

EXAMPLE 3: Geo-query encoded as an HTTP Query String, please note that this is HTTP binding specific.

?type=Vehicle&georel=near;maxDistance==2000&geometry=Point&coordinates=[8,40]

The semantics of the different geospatial relationships defined above is as follows, and shall be supported by compliant implementations:

- **near** statement (production rule named *nearRel*):
  - *maxDistance* modifier. For an entity to match it has to be within the buffer geometric object (as defined by [14]) given by the reference geometry, with distance (in meters) equal to the number conveyed (production rule named *PositiveNumber*).
  - *minDistance* modifier. For an entity to match it has to be disjoint with the buffer geometric object (as defined by [14]) given by the reference geometry, with distance (in meters) equal to the number conveyed (production rule named *PositiveNumber*).
- **equals** relationship (production rule named *equalsRel*). For an entity to match, the target geometry shall be equal, as specified by [14], to the reference geometry.

- **disjoint** relationship (production rule named *disjointRel*). For an entity to match, the target geometry shall be disjoint, as specified by [14], to the reference geometry.
- **intersects** relationship (production rule named *intersectsRel*). For an entity to match, the target geometry shall intersect, as specified by [14], with the reference geometry.
- **within** relationship (production rule named *withinRel*). For an entity to match, the target geometry shall to be within, as specified by [14], the reference geometry.
- **contains** relationship (production rule named *containsRel*). For an entity to match, the target geometry shall contain, as specified by [14], the reference geometry.
- **overlaps** relationship (production rule named *overlapsRel*). For an entity to match, the target geometry shall overlap, as specified by [14], the reference geometry.

When resolving geo-queries, Entities which do not convey the target *GeoProperty* of the query shall be considered as non-matching.

4.11 NGSI-LD Temporal Query language

The NGSI-LD Temporal Query language shall be supported by implementations. It is intended to define predicates which allow testing whether Temporal Properties of NGSI-LD Entities, Properties and Relationships, are within certain temporal constraints. In particular it can be used to request historic Property values and Relationships that were valid within the specified timeframe.

The following grammar defines the syntax that shall be supported:

```
timerel = beforeRel / afterRel / betweenRel
beforeRel = "before"
afterRel = "after"
betweenRel = "between"
```

The points in time for comparison are defined as follows:

- A **time** element, which shall represent the comparison point for the *before* and *after* relation and the starting point for the *between* relation. It shall be represented as *DateTime* (mandated by clause 4.6.3).
- An **endtime** element, which is only used for the *between* relation and shall represent the end point for comparison. It shall be represented as *DateTime* (mandated by clause 4.6.3).

The Temporal Property (see clause 4.8) to which the temporal query is to be applied can be specified by **timeproperty**. If no *timeproperty* is specified, the temporal query is applied to the default Temporal Property *observedAt*.

EXAMPLE 1: timerel=before

time=2017-12-13T14:20:00Z

EXAMPLE 2: timerel=between

time=2017-12-13T14:20:00Z

endtime==2017-12-13T14:40:00Z

timeproperty=modifiedAt

EXAMPLE 3: Temporal query encoded as HTTP Query String, please note that this is HTTP binding specific.

?type=Vehicle&timerel=between&time=2017-12-13T14:20:00Z&timeproperty=observedAt

The semantics of the different temporal relations defined above is as follows, and shall be supported by compliant implementations:

- **before** relationship (production rule named *beforeRel*). For a Temporal Property to match, the value of the specified Temporal Property (or *observedAt* as default) has to be before the time specified by *time*;

- **after** relationship (production rule named *afterRel*). For a Temporal Property to match, the value of the specified Temporal Property (or *observedAt* as default) has to be after the time specified by *time*;
- **between** relationship (production rule named *betweenRel*). For a Temporal Property to match, the value of the specified Temporal Property (or *observedAt* as default) has to be after the time specified by *time* and before the time specified by *endtime*.

When resolving temporal queries, Entities which do not convey the target Temporal Property of the query shall be considered as non-matching.

4.12 NGSI-LD Query pagination

NGSI-LD Query operations can potentially return a result set including a large number of NGSI-LD Elements, so that pagination of query results shall be supported by compliant implementations. Nonetheless, the NGSI-LD API is agnostic about specific pagination mechanisms and only defines the behaviour that shall be observed by NGSI-LD Systems. Facebook™ for Developers (https://developers.facebook.com/docs/graph-api/using-graph-api/#paging) describes different pagination mechanisms that can be of help when it comes to the implementation of NGSI-LD Query Pagination.

For each Query operation, NGSI-LD Systems shall:

- provide a mechanism to iterate through the NGSI-LD Elements of a result set without exhausting NGSI-LD Client or Broker resources;
- provide a mechanism to flag NGSI-LD Clients when there are remaining NGSI-LD Elements to be traversed as part of a result set;
- allow NGSI-LD Clients specifying a limit (page size), as a parameter of API Query operations, to the number of NGSI-LD Elements (at a maximum) retrieved by the implementation for each pagination iteration;
- define a **default limit** (default page size) to the number of NGSI-LD Elements retrieved per pagination iteration;
- allow NGSI-LD Clients iterating forwards and backwards through a result set.

NGSI-LD implementations should:

- avoid Denial of Service attacks or other potential security risks, by defining a hard limit to the size of generated response payload body while paginating. For instance, certain queries can be rejected by issuing an error of type *TooManyResults*.

NGSI-LD implementations may:

- warn NGSI-LD Clients when result sets become invalid due to dynamic changes in NGSI-LD Elements (additions, deletions) occurred while iterating over pages.

The concrete realization of the features described above might depend on each API binding. Nonetheless, NGSI-LD Systems shall implement pagination features as mandated by the present clause, for any API binding.

5 API Operation Definition

5.1 Introduction

This clause defines data structures and operations of the NGSI-LD API. No specific binding is assumed. Clause 6 maps these operations and data types to the HTTP REST binding.

NOTE: In UML diagrams dotted arrows denote a response to a request.

5.2 Data types

5.2.1 Introduction

Implementations shall support the data types defined by the clauses below. For each member defined by each data type (including nested ones) a term shall be added to the Core @context, as mandated by clause 4.5.

None of the members described admit a *null* value, except when they are used in the context of an update operation (see clause 5.5.8) and implementations shall raise an error of type *BadRequestData* if a *null* value is encountered.

Non-normative JSON Schema [i.11] definitions of the referred data types are also available at [i.13].

5.2.2 Common members

The JSON-LD representation of NGSI-LD Entity, Property, Relationship, Context Source Registration and Subscription can include the common members described by table 5.2.2-1.

Those members are read-only, and shall be automatically generated by NGSI-LD implementations. They shall not be provided by Context Producers. In the event that they are provided (in update or create operations) NGSI-LD implementations shall ignore them.

In query or retrieve operations implementations shall only generate common members (table 5.2.2-1) when the Context Consumer explicitly asks for their inclusion. Clause 6.3.11 defines the mechanism offered by the HTTP binding for such purpose.

Table 5.2.2-1: Common members of NGSI-LD elements

Name	Data type	Restriction	Cardinality	Description
createdAt	string	*DateTime* (clause 4.6.3)	0..1	Entity creation timestamp. See clause 4.8
modifiedAt	string	*DateTime* (clause 4.6.3)	0..1	Entity last modification timestamp. See clause 4.8

5.2.3 @context

When encoding NGSI-LD Entities, Context Source Registrations, Subscriptions and Notifications, as pure JSON-LD (MIME type "application/ld+json"), a proper @context shall be included as a special member of the corresponding JSON-LD Object. Table 5.2.3-1 gives a precise definition of this special member.

Table 5.2.3-1: JSON-LD @context tagged member

Name	Data type	Restriction	Cardinality	Description
@context	URI, JSON Object, or JSON Array	See [2], section 5.1.	0..1	JSON-LD @context.

5.2.4 Entity

This type represents the data needed to define an NGSI-LD entity as mandated by clause 4.5.

The supported JSON members shall follow the requirements provided in table 5.2.4-1.

Table 5.2.4-1: NGSI-LD Entity data type definition

Name	Data type	Restriction	Cardinality	Description
id	URI		1	Entity id
type	URI or String	Entity Type	1	Entity Type. Both short hand string (type name) or URI are allowed
location	GeoProperty	See datatype definition on clause 5.2.7	0..1	Default geospatial Property of an entity. See clause 4.7
observationSpace	GeoProperty		0..1	See clause 4.7
operationSpace	GeoProperty		0..1	See clause 4.7
<Property Name>	Property	See datatype definition on clause 5.2.5	0..N	Property as mandated by clause 4.5.1
<Relationship Name>	Relationship	See datatype definition on clause 5.2.6	0..N	Relationship as mandated by clause 4.5.2

5.2.5 Property

This type represents the data needed to define a Property as mandated by clause 4.5.1.

The supported JSON members shall follow the requirements provided in table 5.2.5-1.

Table 5.2.5-1: NGSI-LD Property data type definition

Name	Data type	Restriction	Cardinality	Description
type	string	It shall be equal to "Property"	1	Node type
value	Any JSON value as defined by IETF RFC 8259 [6]	See NGSI-LD Value definition at clause 3.1	1	Property Value
observedAt	string	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
unitCode	string	As mandated by [15]	0..1	Property Value's unit code
datasetId	URI		0.1	It allows identifying a set or group of property values
<Property Name>	Property		0..N	Properties of Property
<Relationship Name>	Relationship	See datatype definition on clause 5.2.6	0..N	Relationships of Property

5.2.6 Relationship

This type represents the data needed to define a Relationship as mandated by clause 4.5.2.

The supported JSON members shall follow the requirements provided in table 5.2.6-1.

Table 5.2.6-1: NGSI-LD Relationship data type definition

Name	Data type	Restriction	Cardinality	Description
type	string	It shall be equal to "Relationship"	1	Node type
object	URI		1	Relationship's target object
observedAt	string	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
datasetId	URI		0..1	It allows identifying a set or group of target relationship objects
<Property Name>	Property	See datatype definition on clause 5.2.5	0..N	Properties of the Relationship
<Relationship Name>	Relationship		0..N	Relationships of the Relationship

5.2.7 GeoProperty

This type represents the data needed to define a *GeoProperty*.

The supported JSON members shall follow the requirements provided in table 5.2.7-1.

Table 5.2.7-1: NGSI-LD GeoProperty data type definition

Name	Data type	Restriction	Cardinality	Description
type	string	It shall be equal to "GeoProperty"	1	Node type
value	JSON Object	As mandated by clause 4.7	1	Geolocation encoded as GeoJSON [8]
observedAt	string	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
datasetId	URI		0..1	It allows identifying a set or group of target relationship objects. See clause 4.5.5
<Property Name>	Property		0..N	Properties of Property
<Relationship Name>	Relationship	See datatype definition on clause 5.2.6	0..N	Relationships of Property

5.2.8 EntityInfo

This type represents what Entities, Entity Types or group of Entity ids (as a regular expression pattern mandated by [11]) can be provided (by Context Sources) or Subscribed to by Context Consumers.

The JSON members shall follow the indications provided in table 5.2.8-1. *id* takes precedence over *idPattern*.

Table 5.2.8-1: EntityInfo data type definition

Name	Data type	Restrictions	Cardinality	Description
id	string	valid URI	0..1	Entity identifier
idPattern	string	Regular expression as per IEEE POSIX 1003.2™ [11]	0..1	A regular expression which denotes a pattern that shall be matched by the provided or subscribed Entities
type	URI or String	Fully Qualified Name of an Entity Type or the Entity Type Name as a short-hand string. See clause 4.6.2	1	Entity Type

5.2.9 CsourceRegistration

This type represents the data needed to register a new Context Source.

The supported JSON members shall follow the indications provided in table 5.2.9-1.

Table 5.2.9-1: CsourceRegistration data type definition

Name	Data type	Restriction	Cardinality	Description
id	URI	At creation time, If it is not provided, it will be assigned during registration process and returned to client. It cannot be later modified in update operations	0..1	Unique registration identifier. (JSON-LD @id). There may be multiple registrations per Context Source, i.e. the id is unique per registration
type	string	"ContextSource Registration"	1	JSON-LD @type Use reserved type for identifying Context Source Registration
name	string	Non-empty string	0..1	A name given to this Context Source Registration

Name	Data type	Restriction	Cardinality	Description
description	string	Non-empty string	0..1	A description of this Context Source Registration
information	RegistrationInfo[]	See data type definition in clause 5.2.10. Empty array (0 length) is not allowed	1	Describes the Entities, Properties and Relationships for which the Context Source may be able to provide information
observationInterval	TimeInterval	See data type definition in clause 5.2.11	0..1	If present, the Context Source can be queried for Temporal Entity Representations. (If latest Entity information is also provided, a separate Context Registration is needed for this purpose). The *observationInterval* specifies the time interval for which the Context Source can provide Entity information as specified by the *observedAt* Temporal Property. A temporal query based on the *observedAt* Temporal Property, which is the default, is matched against the *observationInterval* for overlap
managementInterval	TimeInterval	See data type definition in clause 5.2.11	0..1	If present, the Context Source can be queried for Temporal Entity Representations. (If latest Entity information is also provided, a separate Context Registration is needed for this purpose). The *managementInterval* specifies the time interval for which the Context Source can provide Entity information as specified by the *createdAt* and *modifiedAt* Temporal Properties. A temporal query based on the *createdAt* or *modifiedAt* Temporal Property is matched against the *managementInterval* for overlap
location	GeoJSON Geometry as mandated by clause 4.7		0..1	Location for which the Context Source may be able to provide information
observationSpace	GeoJSON Geometry as mandated by clause 4.7		0..1	Geographic location that includes the observation spaces of all entities as specified by their respective *observationSpace GeoProperty* for which the Context Source may be able to provide information
operationSpace	GeoJSON Geometry as mandated by clause 4.7		0..1	Geographic location that includes the operation spaces of all entities as specified by their respective *operationSpace GeoProperty* for which the Context Source may be able to provide information

Name	Data type	Restriction	Cardinality	Description
expires	string	*DateTime* (clause 4.6.3)	0..1	Provides an expiration date. When passed the Context Source Registration will become invalid and the Context Source might no longer be available
endpoint	URI	It shall be a dereferenceable URI	1	Endpoint expressed as dereferenceable URI through which the Context Source exposes its NGSI-LD interface
<Csource Property Name>	Any JSON value as defined by [6]		0..N	Each Context Source Property pertains to a characteristic of the Context Source the Context Source Registration describes

5.2.10 RegistrationInfo

The supported JSON members shall follow the requirements provided in table 5.2.10-1.

Table 5.2.10-1: RegistrationInfo data type definition

Name	Data type	Restrictions	Cardinality	Description
entities	EntityInfo []	See data type definition on clause 5.2.8. Empty array (0 length) is not allowed	0..1	Describes the entities for which the CSource may be able to provide information
properties	string []	Property Name as short-hand string. Empty array is not allowed	0..1	Describes the Properties that the CSource may be able to provide
relationships	string []	Relationship Name as short-hand string. Empty array is not allowed	0..1	Describes the Relationships that the CSource may be able to provide

At least one element of *RegistrationInfo* shall be present.

5.2.11 TimeInterval

The supported JSON members shall follow the requirements provided in table 5.2.11-1.

Table 5.2.11-1: TimeInterval data type definition

Name	Data type	Restrictions	Cardinality	Description
start	string	*DateTime* (clause 4.6.3)	1	Describes the start of the time interval
end	string	*DateTime* (clause 4.6.3)	0..1	Describes the end of the time interval. If not present the interval is open

5.2.12 Subscription

This datatype represents a Context Subscription.

The supported JSON members shall follow the requirements provided in table 5.2.12-1.

Table 5.2.12-1: Subscription data type definition

Name	Data type	Restrictions	Cardinality	Description
id	URI	At creation time, If it is not provided, it will be assigned during subscription process and returned to client. It cannot be later modified in update operations	0..1	Subscription identifier (JSON-LD @id)
type	string	It shall be equal to "Subscription"	1	JSON-LD @type
name	string		0..1	A (short) name given to this Subscription
description	string		0..1	Subscription description
entities	EntityInfo[]	See data type definition on clause 5.2.8. Empty array (0 length) is not allowed	0..1	Entities subscribed
watchedAttributes	string[]	Attribute Name as short-hand string. if *timeInterval* is present it shall not appear (0 cardinality). Empty array (0 length) is not allowed	0..1	Watched Attributes (Properties or Relationships). If not defined it means any Attribute
timeInterval	Number	Greater than 0 if *watchedAttributes* is present it shall not appear (0 cardinality)	0..1	Indicates that a notification shall be delivered periodically regardless of attribute changes. Actually, when the time interval (in seconds) specified in this value field is reached
q	string	A valid query string as per clause 4.9	0..1	Query that shall be met by subscribed entities in order to trigger the notification
geoQ	GeoQuery	See data type definition on clause 5.2.13	0..1	Geo-Query that shall be met by subscribed entities in order to trigger the notification
csf	string	A valid query string as per clause 4.9	0..1	Context source filter that shall be met by Context Source Registrations describing Context Sources to be used for Entity Subscriptions
isActive	boolean	True by default	0..1	Allows clients to temporarily pause the subscription by making it inactive. *True* indicates that the Subscription is under operation. *False* indicates that the subscription is paused and notifications shall not be delivered
notification	NotificationParams	See data type definition on clause 5.2.14	1	Notification details
expires	string	*DateTime* (see clause 4.6.3)	0..1	Expiration date for the subscription
throttling	Number	Greater than 0. If *timeInterval* is present it shall not appear (0 cardinality)	0..1	Minimal period of time in seconds which shall elapse between two consecutive notifications
temporalQ	TemporalQuery	See data type definition on clause 5.2.21	0..1	Temporal Query to be used *only in Context Registration Subscriptions* for matching Context Source Registrations of Context Sources providing temporal information

At least one of (a) *entities* or (b) *watchedAttributes* shall be present.

The members (defined by table 5.2.12-2) of the *Subscription* data structure are also defined. They are read-only and shall be automatically generated by NGSI-LD implementations. They shall not be provided by Context Subscribers. In the event that they are provided (in update or create operations) NGSI-LD implementations shall ignore them.

Table 5.2.12-2: Additional members of the Subscription data type

Name	Data type	Restrictions	Cardinality	Description
status	string	Allowed values: "active" "paused" "expired"	0..1	Read-only. Provided by the system when querying the details of a subscription

5.2.13 GeoQuery

This datatype represents a geo-query used for Subscriptions.

The supported JSON members shall follow the requirements provided in table 5.2.13-1.

Table 5.2.13-1: GeoQuery data type definition

Name	Data type	Restrictions	Cardinality	Description
geometry	string	A valid GeoJSON [8] geometry type excepting *GeometryCollection*	1	Type of the reference geometry
coordinates	JSON Array or string	A JSON Array coherent with the geometry type as per IETF RFC 7946 [8]	1	Coordinates of the reference geometry. For the sake of JSON-LD compatibility It can be encoded as a string as described in clause 4.7.1
georel	string	A valid geo-relationship as defined by clause 4.10	1	Geo-relationship (near, within, etc.)
geoproperty	string	Attribute Name as short-hand string	0..1	Specifies the GeoProperty to which the GeoQuery is to be applied. If not present, the default GeoProperty is *location*

5.2.14 NotificationParams

5.2.14.1 NotificationParams data type definition

This datatype represents the parameters that allow to convey the details of a notification.

The supported JSON members shall follow the requirements provided in table 5.2.14.1-1.

Table 5.2.14.1-1: NotificationParams data type definition

Name	Data type	Restrictions	Cardinality	Description
attributes	string[]	Attribute Name as short-hand string. Empty array (0 length) is not allowed	0..1	Entity Attribute Names (Properties or Relationships) to be included in the notification payload body. If undefined it will mean all Attributes
format	string	It shall be one of: "keyValues" "normalized"	0..1	Conveys the representation format of the entities delivered at notification time. By default, it will be in normalized format
endpoint	Endpoint	See data type definition on clause 5.2.15	1	Notification endpoint details
status	string	Allowed values: "ok", "failed"	0..1	Status of the Notification. It shall be "ok" if the last attempt to notify the subscriber succeeded. It shall be "failed" if the last attempt to notify the subscriber failed

5.2.14.2 Additional members

The members (defined by table 5.2.14.2-1) of the *NotificationParams* data structure are also defined. They are read-only, and shall be automatically generated by NGSI-LD implementations. They shall not be provided by Context Subscribers. In the event that they are provided (in update or create operations) NGSI-LD implementations shall ignore them.

In query or retrieve operations involving Subscriptions, implementations shall generate them as part of their representation.

Table 5.2.14.2-1: Additional members of the NotificationParams data structure

Name	Data type	Restrictions	Cardinality	Description
timesSent	Number	Greater than 0	0..1	Number of times that the notification was sent. Provided by the system when querying the details of a subscription
lastNotification	string	*DateTime* (clause 4.6.3)	0..1	Timestamp corresponding to the instant when the last notification was sent. Provided by the system when querying the details of a subscription
lastFailure	string	*DateTime* (clause 4.6.3)	0..1	Timestamp corresponding to the instant when the last notification resulting in failure (for instance, in the HTTP binding, an HTTP response code different than 200) was sent. Provided by the system when querying the details of a subscription
lastSuccess	string	*DateTime* (clause 4.6.3)	0..1	Timestamp corresponding to the instant when the last successful (200 OK response) notification was sent. Provided by the system when querying the details of a subscription

5.2.15 Endpoint

This datatype represents the parameters that are required in order to define an endpoint for notifications.

The supported JSON members shall follow the indications provided in table 5.2.15-1.

Table 5.2.15-1: Endpoint data type definition

Name	Data type	Restrictions	Cardinality	Description
uri	URI	Dereferenceable URI	1	URI which conveys the endpoint which will receive the notification
accept	string	MIME type. It shall be one of: "application/json" "application/ld+json"	0..1	Intended to convey the MIME type of the notification payload body (JSON or JSON-LD)

5.2.16 BatchOperationResult

This datatype represents the result of a batch operation.

The supported JSON members shall follow the indications provided in table 5.2.16-1.

Table 5.2.16-1: BatchOperationResult data type definition

Name	Data type	Restrictions	Cardinality	Description
success	URI[]	Entity Id. Empty Array if no Entity was successfully treated	1	Array of Entity Ids corresponding to the Entities that were successfully treated by the concerned operation
errors	BatchEntityError[]	Empty Array if no errors happened	1	One array item per Entity in error

5.2.17 BatchEntityError

This datatype represents an error raised (associated to a particular Entity) during the execution of a batch operation.

The supported JSON members shall follow the indications provided in table 5.2.17-1.

Table 5.2.17-1: BatchEntityError data type definition

Name	Data type	Restrictions	Cardinality	Description
entityId	URI	Entity Id	1	Entity Id corresponding to the Entity in error
error	ProblemDetails [10]		1	One instance per Entity in error

5.2.18 UpdateResult

This datatype represents the result of Attribute update (append or update) operations in the NGSI-LD API.

The supported JSON members shall follow the indications provided in table 5.2.18-1.

Table 5.2.18-1: UpdateResult data type definition

Name	Data type	Restrictions	Cardinality	Description
updated	string[]		1	List of Attributes (represented by their Name) that were appended or updated.
notUpdated	NotUpdatedDetails[]	See clause 5.2.19	1	List which contains the Attributes (represented by their Name) that were not updated, together with the reason for not being updated.

5.2.19 NotUpdatedDetails

This datatype represents additional information provided by an implementation when an Attribute update did not happen. See also clause 5.2.18.

The supported JSON members shall follow the indications provided in table 5.2.19-1.

Table 5.2.19-1: NotUpdatedDetails data type definition

Name	Data type	Restrictions	Cardinality	Description
attributeName	string		1	Attribute name
reason	string		1	Reason for not having changed such Attribute

5.2.20 EntityTemporal

This is the same data type as mandated by clause 5.2.4 with the only deviation that the representation of Properties and Relationships shall be the temporal one (arrays of (Property or Relationship) instances represented by JSON-LD objects).

5.2.21 TemporalQuery

This datatype represents a temporal query.

The supported JSON members shall follow the requirements provided in table 5.2.21-1.

Table 5.2.21-1: TemporalQuery data type definition

Name	Data type	Cardinality	Description
timerel	String representing the temporal relationship as defined by clause 4.11	1	Allowed values: "before", "after" and "between"
time	String representing the *time* parameter as defined by clause 4.11	1	It shall be a *DateTime*
endTime	String representing the *endTime* parameter as defined by clause 4.11	0..1	It shall be a *DateTime.* Cardinality shall be 1 if *timerel* is equal to "between"
timeproperty	String representing a Property name	0..1	The name of the Property that contains the temporal data that will be used to resolve the temporal query. If not specified, the default is "observedAt"

5.3 Notification data types

5.3.1 Notification

This datatype represents the parameters that allow building a notification to be sent to a subscriber. How to build this notification is detailed in clause 5.8.6.

The supported JSON members shall follow the indications provided in table 5.3.1-1.

Table 5.3.1-1: Notification data type definition

Name	Data type	Restrictions	Cardinality	Description
id	URI		1	Notification identifier (JSON-LD @id). It shall be automatically generated by the implementation
type	String	It shall be equal to "Notification"	1	JSON-LD @type
subscriptionId	URI		1	Identifier of the subscription that originated the notification
notifiedAt	string	*DateTime* (clause 4.6.3)	1	Timestamp corresponding to the instant when the notification was generated by the system
data	NGSI-LD Entity[]		1	The content of the notification as NGSI-LD Entities. See clause 5.2.4

5.3.2 CsourceNotification

This datatype represents the parameters that allow building a Context Source Notification to be sent to a subscriber. How to build this notification is detailed in clause 5.11.7.

The supported JSON members shall follow the indications provided in the table 5.3.2-1.

Table 5.3.2-1: CsourceNotification data type definition

Name	Data type	Restrictions	Cardinality	Description
id	URI		1	Csource notification identifier (JSON-LD @id)
type	string	It shall be equal to "ContextSource Notification"	1	JSON-LD @type
subscriptionId	URI		1	Identifier of the subscription that originated the notification
notifiedAt	string	*DateTime* (see clause 4.6.3)	1	Timestamp corresponding to the instant when the notification was generated by the system
data	Csource Registration[]		1	The content of the notification as NGSI-LD entities. See clause 5.2.4
triggerReason	string	*TriggerReasonEnumeration* (see clause 5.3.3)	1	Indicates whether the Csources in the CsourceRegistration(s) in data are newly matching (initial notification or creation), have been updated (but still match) or do not match any longer

5.3.3 TriggerReasonEnumeration

The enumeration can take one of the following values:

- **"newlyMatching"** - describes the case that the notified Context Source Registration(s) newly match(es) the identified subscription. This value is used in the first notification and whenever a new Context Source Registration matching the Subscription has been registered, or an existing Context Source Registration that did not match before has been updated in such a way that it matches now.
- **"updated"** - describes the case that the notified Context Source Registration that was part of a previous notification has been updated, but still matches the Subscription.
- **"noLongerMatching"** - describes the case that the notified Context Source Registration that was part of a previous notification no longer matches the Subscription, i.e. as a result of an update or because it was deleted.

5.4 NGSI-LD Fragments

When updating NGSI-LD elements (Entities, Context Source Registrations or Context Subscriptions) it is necessary to have a means of describing a set of modifications to their content.

An NGSI-LD Fragment is a JSON merge patch document [16] and [i.10] which describes changes to be made to a target JSON-LD document using a syntax that closely mimics the document being modified.

An NGSI-LD Fragment is a JSON-LD Object which shall include the following members:

- *id* (it could be omitted for certain bindings if it can be determined from the operation signature). It shall be equal to the id of the target (mutated) NGSI-LD element.
- *type* (it could be omitted for certain bindings if it can be determined from the operation signature). It shall be equal to the Type Name of the target NGSI-LD element.
- A member (following the same data representation and nesting structure) for each new member to be added to the target NGSI-LD element.
- A member (following the same data representation and nesting structure) for each new member to be modified in the target NGSI-LD element, which value shall correspond to the new member value to be given.
- A member (following the same data representation and nesting structure) with value equal to *null* for each member to be removed from the target NGSI-LD element.

EXAMPLE: The following NGSI-LD Fragment allows to modify a Context Subscription by changing its endpoint's URI:

```
{
  "id": "urn:ngsi-ld:Subscription:MySubscription",
   "type": "Subscription",
   "endpoint": {
      "uri": "http://example.org/newNotificationEndPoint"
   }
}
```

5.5 Common behaviours

5.5.1 Introduction

This clause defines common behaviours for the API operations.

When comparing URIs, implementations shall follow the recommendations of IETF RFC 3986 [5], section 6.

5.5.2 Error types

Table 5.5.2-1 details a list of error types defined by NGSI-LD. The particular conditions under which error type shall be raised are defined when describing each operation supported by the API.

Table 5.5.2-1: Error types in NGSI-LD

Error Type	Description
https://uri.etsi.org/ngsi-ld/errors/InvalidRequest	The request associated to the operation is syntactically invalid or includes wrong content
https://uri.etsi.org/ngsi-ld/errors/BadRequestData	The request includes input data which does not meet the requirements of the operation
https://uri.etsi.org/ngsi-ld/errors/AlreadyExists	The referred element already exists
https://uri.etsi.org/ngsi-ld/errors/OperationNotSupported	The operation is not supported
https://uri.etsi.org/ngsi-ld/errors/ResourceNotFound	The referred resource has not been found
https://uri.etsi.org/ngsi-ld/errors/InternalError	There has been an error during the operation execution
https://uri.etsi.org/ngsi-ld/errors/TooComplexQuery	The query associated to the operation is too complex and cannot be resolved
https://uri.etsi.org/ngsi-ld/errors/TooManyResults	The query associated to the operation is producing so many results that can exhaust client or server resources. It should be made more restrictive
https://uri.etsi.org/ngsi-ld/errors/LdContextNotAvailable	A remote JSON-LD @context referenced in a request cannot be retrieved by the NGSI-LD Broker and expansion or compaction cannot be performed

5.5.3 Error response payload body

When reporting errors back to clients, NGSI-LD implementations shall generate a JSON object in accordance with IETF RFC 7807 [10], section 3.1, including, at least the following terms:

- **type:** Error type as per clause 5.5.2.
- **title:** Error title which shall be a short string summarizing the error.
- **detail:** A detailed message that should convey enough information about the error.

Even though IETF RFC 7807 [10] defines a specific MIME type for error payloads, NGSI-LD implementations shall use the standard JSON MIME type ("application/json") when reporting errors, so that old clients or existing tools are not broken.

5.5.4 JSON-LD validation

All the operations that take a JSON-LD document as input shall process such JSON-LD document as follows:

- If the request payload body is not a valid JSON document then an error of type *InvalidRequest* shall be raised.
- If the data included by the JSON-LD document is not syntactically correct, according to the @context or the API Data type definitions, then an error of type *BadRequestData* shall be raised. For the avoidance of doubt, this includes validation of member values in accordance with their data type, for instance *DateTime*, and validation of each Relationship's target object which shall be a syntactically valid URI.
- Any attempt to use *null* as member value, with the exception of NGSI-LD Fragments (as mandated by clauses 5.4 and 5.5.8), shall result in an error of type *BadRequestData*.

5.5.5 Default @context assignment

If an input JSON document provided by an API client, does not include an @context and there is no other mechanism available to determine it, then the implementation shall assign the default @context to such JSON document. The default @context shall include all the terms defined by the Core NGSI-LD @context as mandated by clause 4.4.

5.5.6 Operation execution

When executing an operation if an unexpected error happens and the operation cannot be completed, implementations shall raise an error of type *InternalError*. This includes, as well, situations such as database timeouts, etc.

If the NGSI-LD endpoint is not capable of executing the requested operation, an error of type *OperationNotSupported* shall be raised. This may happen in a distributed architecture where a Context Broker might not be able to store Entities (only to forward queries to Context Sources), and, as a result, certain operations such as "Create Entity" might not be supported.

When a query operation is so complex that cannot be resolved by an NGSI-LD system, implementations shall raise an error of type *TooComplexQuery*.

When a query operation is producing so many results that can potentially exhaust client or server resources, or it can be just impractical to be managed, implementations shall raise an error of type *TooManyResults*. The threshold conditions used as criteria to raise such error is up to each implementation.

When a remote JSON-LD @context referenced by an incoming request is not available, implementations shall raise an error of type *LdContextNotAvailable*.

5.5.7 Term to URI expansion or compaction

NGSI-LD API operations allow clients to use short-hand strings as non-qualified names, particularly for Property, Relationship or Type Names. For instance, an API client can refer to the term "Vehicle" as a non-qualified type name. When executing API update-related operations, NGSI-LD systems shall expand terms to URIs, in order to obtain and store Fully Qualified Names. Likewise, when executing query-related operations, NGSI-LD systems shall compact URIs (Fully Qualified Names) to short terms in order to provide short-hand strings to context consumers.

The term to URI expansion or compaction shall be performed using a @context as described by the JSON-LD specification [2], section 5.1. In the absence of a @context, the term expansion or compaction shall be performed using the default @context (clause 5.5.5). For the avoidance of doubt, the @context used to perform compaction or expansion of terms **shall be the one provided by each API call itself** (or the default @context in its absence), and not any other @context which might have been supplied previously. For instance, when querying an Entity, the @context used to perform compaction should be the one provided by the corresponding HTTP request, using the JSON-LD Link header as described by the JSON-LD specification [2], section 5.1.

As the Core @context is protected and cannot be overridden, when performing term to URI expansion or compaction, implementations **shall always consider the Core @context as having absolute precedence**, regardless of the position of the Core @context in the @context array of elements. Nonetheless, NGSI-LD data providers may use appropriate term prefixing to ensure that a proper term to URI expansion or compaction is performed.

At compaction time, in the event that no matching term is found in the current @context, implementations shall render Fully Qualified Names.

EXAMPLE: An entity of type "Vehicle" bound to a certain @context, C, will match a query by "Vehicle" type if and only if the supplied query @context, Q, maps the term "Vehicle" to the same URI as C.

5.5.8 JSON-LD Merge Patch Behaviour

When updating NGSI-LD elements (Entities, Context Source Registrations or Context Subscriptions) using NGSI-LD Fragments, implementations shall determine the exact set of changes being requested by comparing the content of the provided Fragment (patch) against the current content (a JSON-LD object) of the target element.

Implementations shall perform an algorithm equivalent to the one described below (slightly adapted from IETF RFC 7396 [16]), in order to observe the name to URI expansion rules:

- For each member of the Fragment perform the term to URI expansion.
- If the provided Fragment (merge patch) contains members that do not appear within the target (their URIs do not match), those members are added to the target.
- For each member of the Fragment, whose value is different than *null*, contained by the target, the target member value is replaced by value given in the Fragment. In the case of a member representing a Property or Relationship including a *datasetId*, such member is only replaced if the *datasetId* is the same, otherwise the member of the Fragment is added as a new instance to the target.
- For each member of the Fragment, whose value is *null*, contained by the target, the target member is removed. In the case of deleting a specific Entity Attribute, the handling of members with a *datasetId* shall be according to the description in clause 5.6.5. A *datasetId* property cannot be deleted by setting it to the value *null.*

5.5.9 Pagination Behaviour

When resolving NGSI-LD Query operations NGSI-LD Systems shall exhibit the behaviour described by the present clause:

- Let **Md** be equal to the default maximum number of NGSI-LD Elements to be retrieved by the API during each query pagination iteration, as defined by the NGSI-LD implementation.
- Let **Mc** be equal to the maximum number of NGSI-LD Elements to be retrieved as requested by the NGSI-LD Client. If **Mc** is undefined then it shall be equal to **Md**.
- Let **L** be the maximum number of NGSI-LD Elements to be retrieved by the API during each query pagination iteration. **L** shall be equal to **Mc**.
- During query execution and for each pagination iteration, the query resolution mechanisms of the NGSI-LD System shall ensure that only up to a maximum of **L** NGSI-LD Elements are retrieved and returned to the NGSI-LD client, i.e. the maximum page size per iteration shall not overpass **L**. Nonetheless, implementations shall take care of not overpassing a maximum size of response payload body, which, in practice, implies that, under certain circumstances, the number of Elements retrieved per page can be lower than **L**.
- After the retrieval of each page (containing at most **L NGSI-LD Elements**) implementations shall check whether there are pending NGSI-LD Elements to be retrieved in the context of the current query. If that is the case, implementations shall flag NGSI-LD Clients of the existence of such NGSI-LD Elements. Ultimately, the flagging mechanisms used shall depend on each API binding but shall be present as mandated by the present clause.
- When flagging the existence of additional NGSI-LD Elements (pages) pending to be retrieved, generally, implementations shall provide NGSI-LD Clients pointers to get access to both the following page of NGSI-LD Elements and the previous one, according to the current pagination iteration.
- The pointer to the previous page of NGSI-LD Elements shall be included for all pagination iterations excepting the first one, as, obviously, there will be no previous NGSI-LD Elements.

- When the last page of NGSI-LD Elements is reached, only the pointer to the previous page shall be provided to NGSI-LD Clients, so that they can detect that no more NGSI-LD Elements are available.
- The pointers to NGSI-LD Elements shall contain all the parameters needed to allow NGSI-LD Clients to retrieve the next and previous page, without further interactions with the API.

While iterating over a set of pages, there might be changes in the target result set, due to additions or removals of NGSI-LD Elements occurring in between. Implementations may detect those situations and may warn NGSI-LD Clients appropriately.

5.6 Context Information Provision

5.6.1 Create Entity

5.6.1.1 Description

This operation allows creating a new NGSI-LD Entity.

5.6.1.2 Use case diagram

A Context Producer can create an Entity within an NGSI-LD system as shown in figure 5.6.1.2-1.

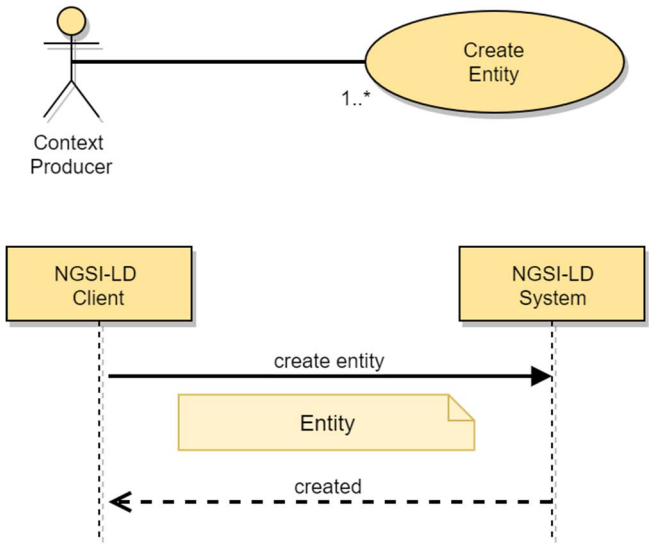

Figure 5.6.1.2-1: Create entity use case

5.6.1.3 Input data

A JSON-LD document representing an NGSI-LD Entity as mandated by clause 5.2.4.

5.6.1.4 Behaviour

Implementations shall exhibit the following behaviour:

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- If the NGSI-LD endpoint already knows about this Entity, because there is an existing entity whose id (URI) is equivalent an error of type *AlreadyExists* shall be raised.
- Otherwise, implementations shall retain the provided entity under the corresponding entity collection.

5.6.1.5 Output data

None.

5.6.2 Update Entity Attributes

5.6.2.1 Description

This operation allows modifying an existing NGSI-LD Entity by updating **already existing** Attributes (Properties or Relationships).

5.6.2.2 Use case diagram

A Context Producer can update Entity Attributes within an NGSI-LD system as shown in figure 5.6.2.2-1.

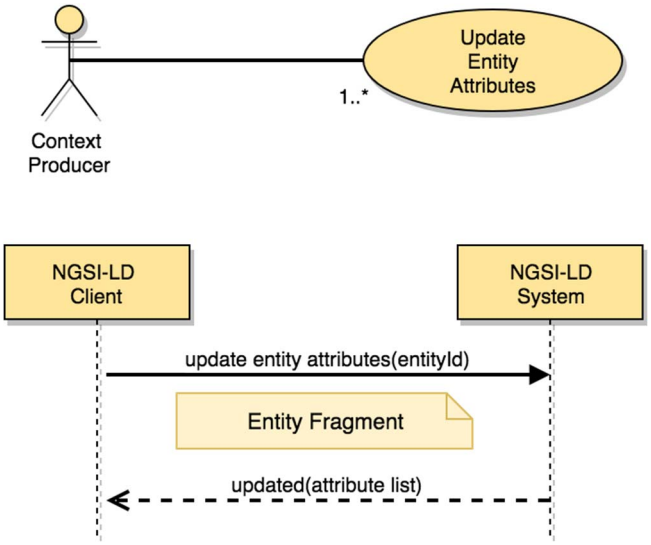

Figure 5.6.2.2-1: Update entity Attributes use case

5.6.2.3 Input data

- A URI representing the id of the Entity to be updated (target Entity).
- A JSON-LD document representing an NGSI-LD Entity Fragment.

5.6.2.4 Behaviour

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent to the target entity, an error of type *ResourceNotFound* shall be raised.
- Execute the behaviour defined on clause 5.5.4 on JSON-LD validation.
- For each of the Attributes included in the Fragment, if the target Entity includes a matching one (considering term expansion rules as mandated by clause 5.5.7), then replace it by the one included by the Fragment. If the Attribute includes a *datasetId*, only an Attribute instance with the same *datasetId* is replaced. In all other cases, the Attribute shall be ignored.

5.6.2.5 Output data

- A status code indicating whether all the new Attributes were updated or only some of them.
- List of Attributes (Properties or Relationships) actually updated.

5.6.3 Append Entity Attributes

5.6.3.1 Description

This operation allows modifying an NGSI-LD Entity by adding new attributes (Properties or Relationships).

5.6.3.2 Use case diagram

A Context Producer can append new Attributes to an existing Entity within an NGSI-LD system as shown in figure 5.6.3.2-1.

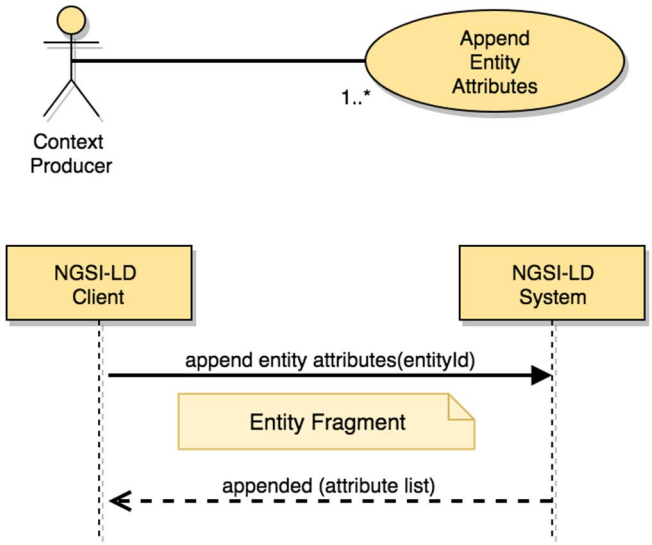

Figure 5.6.3.2-1: Append Entity Attributes use case

5.6.3.3 Input data

- A URI representing the id of the E to be modified (target Entity).
- A JSON-LD document representing an NGSI-LD Entity Fragment.
- An optional flag indicating whether the append operation should overwrite or not existing Attributes. By default, Attributes will be overwritten.

5.6.3.4 Behaviour

The following behaviour shall be exhibited by compliant implementations:

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about this Entity, because there is no an existing Entity which id (URI) is equivalent to the one passed as parameter, an error of type *ResourceNotFound* shall be raised.
- The behaviour defined on clause 5.5.4 on JSON-LD validation.

- For each Attribute (Property or Relationship) included by the Entity Fragment at root level:
  - If the target Entity does not include a matching Attribute (considering term expansion rules as mandated by clause 5.5.7) then such Attribute shall be appended to the target Entity.
  - If the target Entity already includes a matching Attribute (considering term expansion rules as mandated by clause 5.5.7):
    - If a *datasetId* is present in the Attribute included by the Entity Fragment:
      - If an Attribute instance in the target Entity has the same *datasetId*:
        - If overwrite is allowed, then the existing Attribute with the specified *datasetId* in the target Entity shall be replaced by the new one supplied.
        - If overwrite is not allowed the existing Attribute with the specified *datasetId* in the target Entity shall be left untouched.
      - Otherwise the Attribute instance with the specified *datasetId* shall be appended to the target Entity.
    - If no *datasetId* is present in the Attribute included by the Entity Fragment:
      - If overwrite is allowed, then the existing Attribute in the target Entity shall be replaced by the new one supplied.
      - If overwrite is not allowed the existing Attribute in the target Entity shall be left untouched.

5.6.3.5 Output data

- A status code indicating whether all the new Attributes were appended or only some of them.
- List of Attributes (Properties and/or Relationships) actually appended.

5.6.4 Partial Attribute update

5.6.4.1 Description

This operation allows performing a **partial update on an NGSI-LD Entity's Attribute** (Property or Relationship). A partial update only changes the elements provided in an Entity Fragment, leaving the rest as they are.

5.6.4.2 Use case diagram

A Context Producer can carry out a partial Attribute update of an Entity within an NGSI-LD System as shown in figure 5.6.4.2-1.

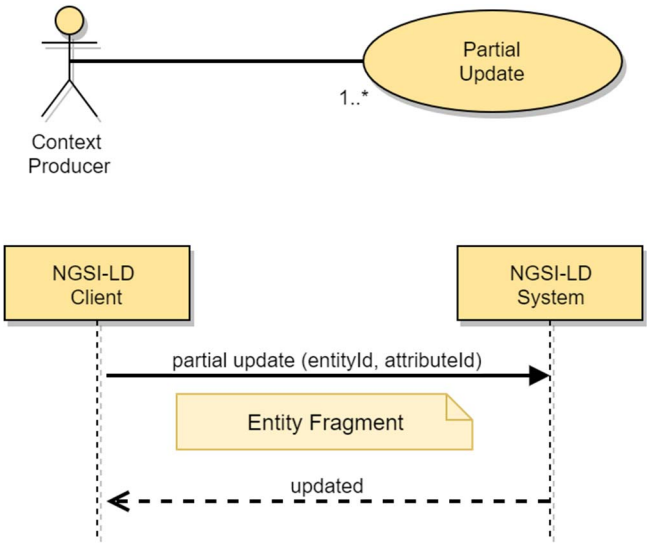

Figure 5.6.4.2-1: Partial Attribute update use case

5.6.4.3 Input data

- Entity Id (URI) of the concerned Entity, the target Entity.
- Target Attribute (Property or Relationship) to be modified, identified by a name.
- A JSON-LD document representing an NGSI-LD Entity Fragment.

5.6.4.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute Name is not valid or it is not present, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no an existing Entity which id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Apply term expansion as mandated by clause 5.5.7, so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- Perform a partial update on the target Attribute following the algorithm mandated by clause 5.5.8.

5.6.4.5 Output data

None.

5.6.5 Delete Entity Attribute

5.6.5.1 Description

This operation allows deleting an NGSI-LD Entity's Attribute (Property or Relationship). The Attribute itself and all its children elements shall be deleted.

5.6.5.2 Use case diagram

A Context Producer can delete a specific Entity Attribute within an NGSI-LD system as shown in figure 5.6.5.2-1.

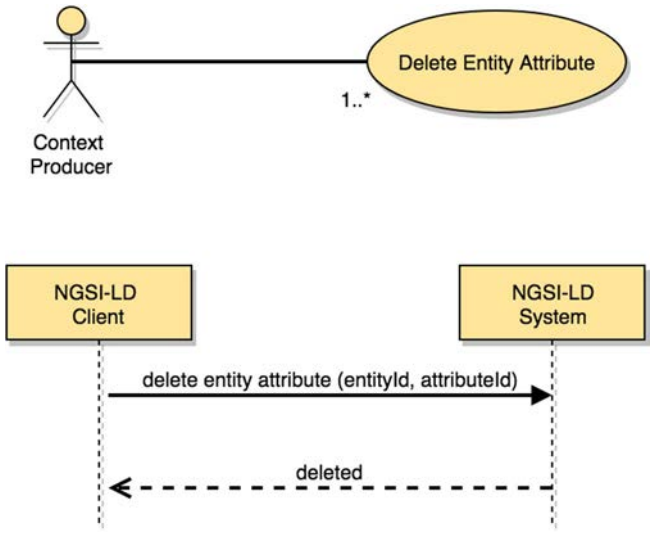

Figure 5.6.5.2-1: Delete Entity Attribute use case

5.6.5.3 Input data

- Entity id (URI) of the concerned Entity, the target Entity.
- Target Attribute (Property or Relationship) to be deleted, identified by a Name.
- An optional parameter identifying the *datasetId* of the target Attribute instance to be deleted.
- An optional flag "deleteAll" indicating whether also all target Attribute instances with a *datasetId* are to be deleted.
- An optional JSON-LD @context.

5.6.5.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name or it is not present, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If the "deleteAll" flag is set, remove all target Attribute instances from the target Entity.
- Otherwise, if a *datasetId* parameter is provided, remove any target Attribute instance from the given dataset only.
- Otherwise remove any target Attribute instance from the target Entity that does not have a *datasetId* set.

5.6.5.5 Output data

None.

5.6.6 Delete Entity

5.6.6.1 Description

This operation allows deleting an NGSI-LD Entity.

5.6.6.2 Use case diagram

A Context Producer can completely delete an Entity within an NGSI-LD system as shown in figure 5.6.6.2-1.

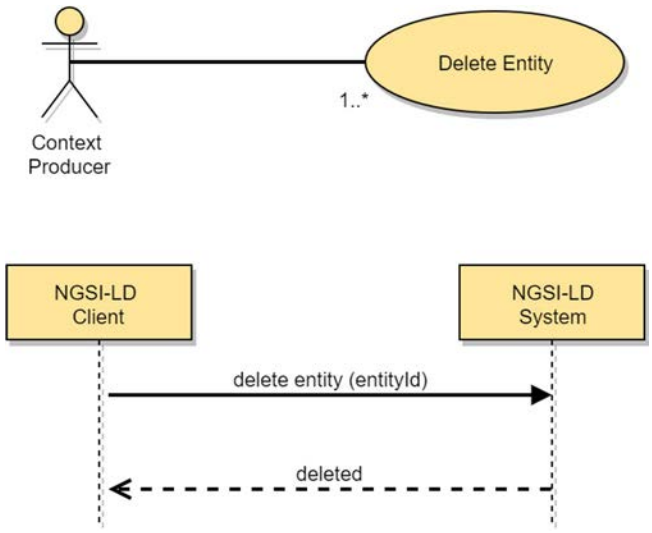

Figure 5.6.6.2-1: Delete Entity use case

5.6.6.3 Input data

- Entity Id (URI) of the Entity to be deleted, the target Entity.

5.6.6.4 Behaviour

- If the target Entity id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, then an error of type *ResourceNotFound* shall be raised.
- Otherwise the Entity shall be removed from the Entity collection.

5.6.6.5 Output data

None.

5.6.7 Batch Entity Creation

5.6.7.1 Description

This operation allows creating a batch of NGSI-LD Entities.

5.6.7.2 Use case diagram

A Context Producer can create a batch of NGSI-LD Entities within an NGSI-LD system as shown in figure 5.6.7.2-1.

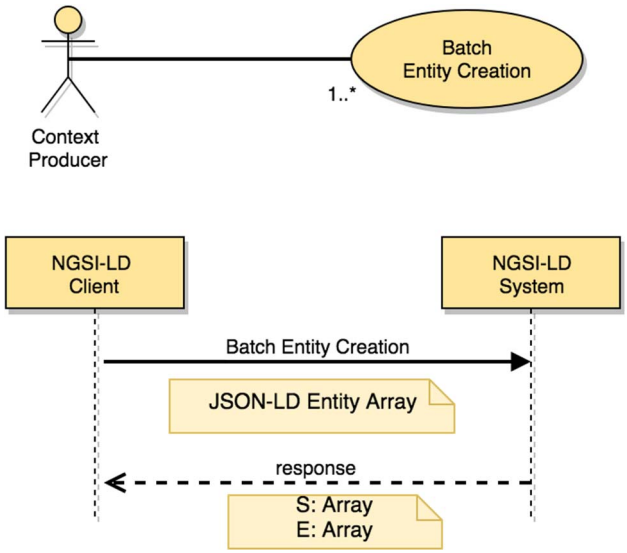

Figure 5.6.7.2-1: Create a batch of Entities use case

5.6.7.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an NGSI-LD Entity as mandated by clause 5.2.4.

5.6.7.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items an error of type *BadRequestData* shall be raised.
- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity successfully created. S shall be initialized to the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized to the empty array.
- For each of the NGSI-LD Entities included in the input Array execute the behaviour defined by clause 5.6.1 as follows:
  - If the Entity was successfully created, then add the corresponding Entity Id to the S array.
  - If the Entity creation failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the related *ProblemDetails*.

5.6.7.5 Output data

- The list of Entities successfully created (S Array).
- The list of Entities in error (E Array).

5.6.8 Batch Entity Creation or Update (Upsert)

5.6.8.1 Description

This operation allows creating a batch of NGSI-LD Entities, updating each of them if they already existed. In some database jargon this kind of operation is known as "upsert".

5.6.8.2 Use case diagram

A Context Producer can create or update a batch of Entities within an NGSI-LD system as shown in figure 5.6.8.2-1.

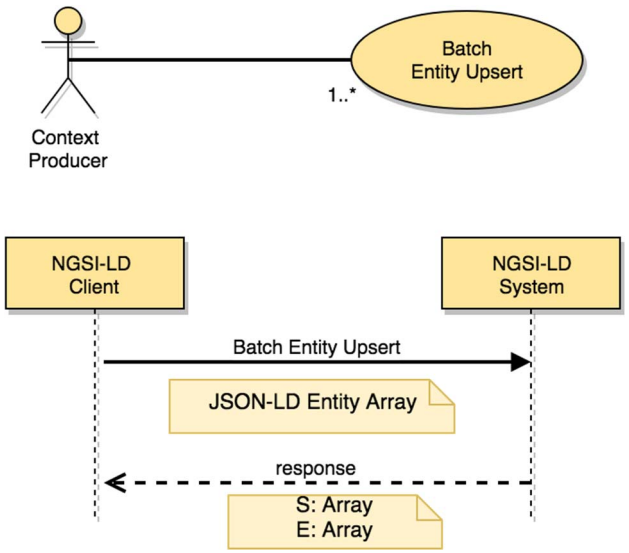

Figure 5.6.8.2-1: Upsert a batch of Entities use case

5.6.8.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an Entity as mandated by clause 5.2.4.
- An optional flag indicating the update mode (only applies in case the Entity already exists):
  - Replace. All the existing Entity content shall be replaced (default mode).
  - Update. Existing Entity content shall be updated.

5.6.8.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items, an error of type *BadRequestData* shall be raised.
- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized to the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized to the empty array.
- For each of the NGSI-LD Entities included in the input Array implementations shall:
  - Retain the concerned Entity under the corresponding Entity collection if it does not exist yet (i.e. no Entity with the same Entity Id is present).

- If there were an existing Entity with the same Entity Id, it shall be completely replaced by the new Entity content provided, if the requested update mode is 'replace'.
- If there were an existing Entity with the same Entity Id, it shall be executed the behaviour defined by clause 5.6.3, if the requested update mode is 'update'.

- If while processing an Entity there is any kind of error or abnormal situation, a *BatchEntityError* shall be added to E containing the failed Entity Id and the related *ProblemDetails*.

5.6.8.5 Output data

- The list of Entities successfully processed (S Array).
- The list of Entities in error (E Array).

5.6.9 Batch Entity Update

5.6.9.1 Description

This operation allows updating a batch of NGSI-LD Entities.

5.6.9.2 Use case diagram

A Context Producer can update a batch of Entities within an NGSI-LD system as shown in figure 5.6.9.2-1.

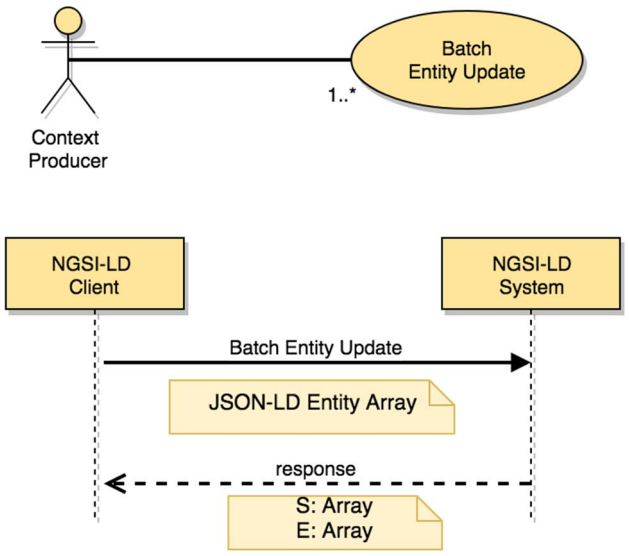

Figure 5.6.9.2-1: Update a batch of Entities use case

5.6.9.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an Entity as mandated by clause 5.2.4.
- An optional flag indicating whether Attributes shall be overwritten or not. By default, Attributes will be overwritten.

5.6.9.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items, an error of type *BadRequestData* shall be raised.

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized as the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized as the empty array.
- For each of the NGSI-LD Entities included in the input Array execute the behaviour defined by clause 5.6.3 as follows:
  - If the Entity was successfully updated (Attributes appended), then add the corresponding Entity Id to the S array.
  - If the Entity update failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the *ProblemDetails* associated.

5.6.9.5 Output data

- The list of Entities successfully processed (S Array).
- The list of Entities in error (E Array).

5.6.10 Batch Entity Delete

5.6.10.1 Description

This operation allows deleting a batch of NGSI-LD Entities.

5.6.10.2 Use case diagram

A Context Producer can delete a batch of Entities within an NGSI-LD system as shown in figure 5.6.10.2-1.

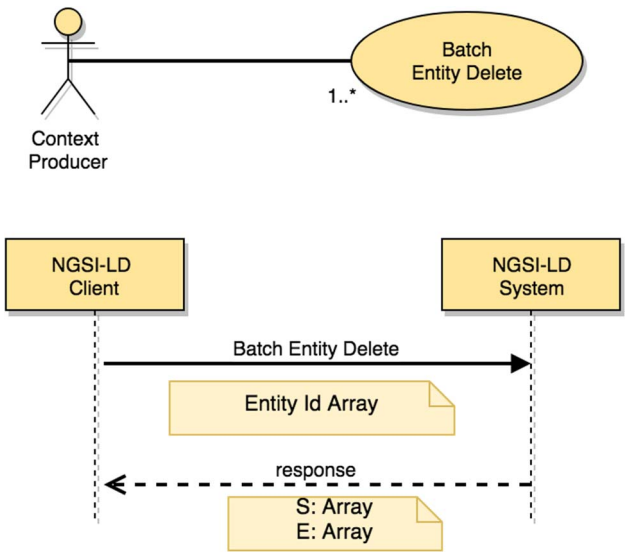

Figure 5.6.10.2-1: Delete a batch of Entities use case

5.6.10.3 Input data

- A JSON-LD Array containing a list of Entity Ids (URIs) that are requested to be deleted.

5.6.10.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items, an error of type *BadRequestData* shall be raised.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized to the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized to the empty array.
- For each of the NGSI-LD Entity Ids included in the input Array execute the behaviour defined by clause 5.6.6 as follows:
  - If the Entity corresponding to an Entity Id was successfully deleted, then add such Entity Id to the S array.
  - If the Entity deletion failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the related *ProblemDetails*.

5.6.10.5 Output data

- The list of Entities successfully processed (S Array).
- The list of Entities in error (E Array).

5.6.11 Create or Update Temporal Representation of an Entity

5.6.11.1 Description

This operation allows creating or updating (by adding new Attribute instances) a Temporal Representation of an Entity.

5.6.11.2 Use case diagram

A Context Producer can create a Temporal Representation of an Entity within an NGSI-LD system as shown in figure 5.6.11.2-1.

Similarly, if the Entity already exists then an Update scenario will be in place.

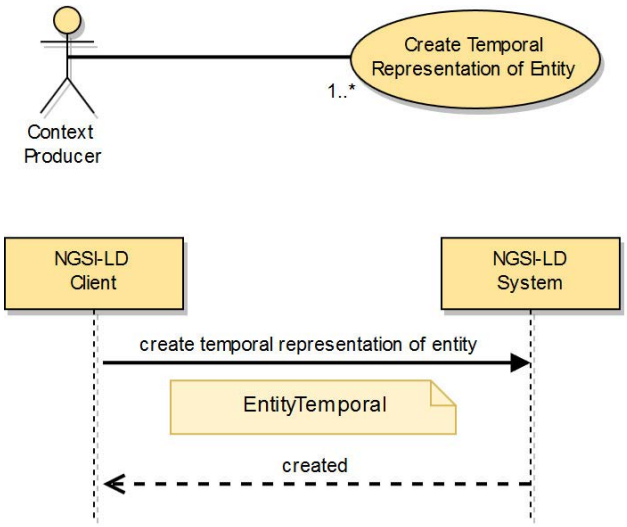

Figure 5.6.11.2-1: Create Temporal Representation of Entity use case

5.6.11.3 Input data

A JSON-LD document representing a Temporal Representation of an Entity as mandated by clause 5.2.20.

5.6.11.4 Behaviour

Implementations shall exhibit the following behaviour:

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- If the NGSI-LD endpoint already knows about this Temporal Representation of an Entity, because there is an existing Temporal Representation of an Entity whose id (URI) is equivalent, then all the Attribute instances included by the Temporal Representation shall be added to the existing Entity as mandated by clause 5.6.12.
- Otherwise, implementations shall retain the provided Temporal Representation of an Entity under the corresponding entity collection.

5.6.11.5 Output data

None.

5.6.12 Add Attributes to Temporal Representation of an Entity

5.6.12.1 Description

This operation allows modifying a Temporal Representation of an Entity by adding new Attribute instances.

5.6.12.2 Use case diagram

A Context Producer can add new Attributes or Attribute instances to an existing Temporal Representation of an Entity within an NGSI-LD system as shown in figure 5.6.12.2-1.

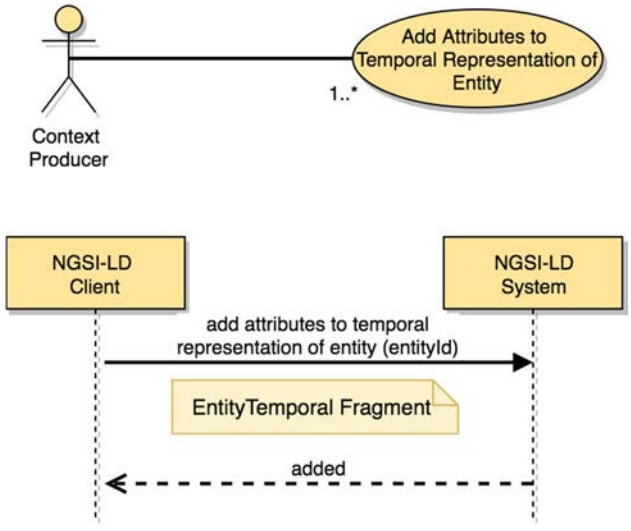

Figure 5.6.12.2-1: Add Attributes to Temporal Representation of Entity use case

5.6.12.3 Input data

- Entity id (URI) which Temporal Representation is to be modified with additional Attributes (target Entity).
- A JSON-LD document representing an NGSI-LD Fragment of *EntityTemporal*, including only the new Attribute instance(s), and contained by an Array.

5.6.12.4 Behaviour

The following behaviour shall be exhibited by compliant implementations:

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the Temporal Representation of the target Entity, because there is no existing Temporal Representation of an Entity whose id (URI) is equivalent to the one passed as parameter, an error of type *ResourceNotFound* shall be raised.
- The behaviour defined in clause 5.5.4 on JSON-LD validation.
- For each Attribute (Property or Relationship) instance included by the *EntityTemporal* Fragment at root level:
  - The Attribute (considering term expansion rules as mandated by clause 5.5.7) instance(s) shall be added to the target Entity. For the avoidance of doubt, if no previous Attribute instances existed, then a new Attribute instance collection shall be created and added to the Entity.

5.6.12.5 Output data

None.

5.6.13 Delete Attribute from Temporal Representation of an Entity

5.6.13.1 Description

This operation allows deleting an Attribute (Property or Relationship) of the Temporal Representation of an Entity. The Attribute itself and all its child NGSI-LD elements shall be deleted.

5.6.13.2 Use case diagram

A Context Producer can delete a specific Attribute of a Temporal Representation of an Entity within an NGSI-LD system as shown in figure 5.6.13.2-1.

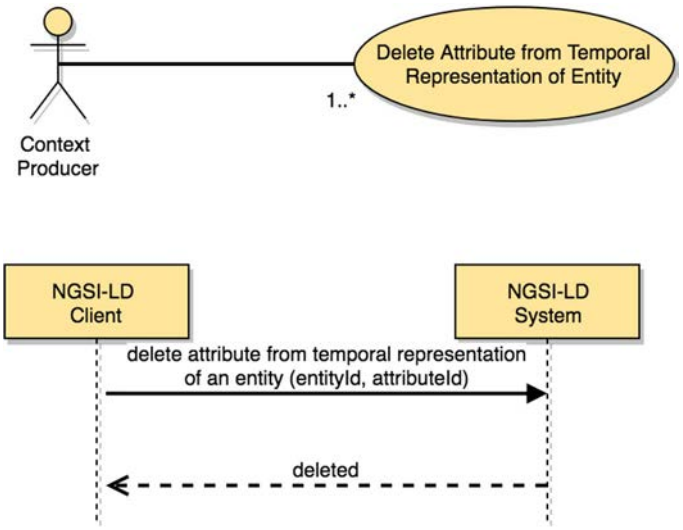

Figure 5.6.13.2-1: Delete Attribute from Temporal Representation of Entity use case

5.6.13.3 Input data

- Entity id (URI) of the target Entity which Temporal Representation is to be modified.
- Target Attribute (Property or Relationship) to be deleted, identified by a Name.
- An optional parameter identifying the dataset (*datasetId*) of the target Attribute instance to be deleted.

- An optional parameter, a flag, (*deleteAll*) indicating whether all target Attribute instances are to be deleted, regardless of *datasetId*.
- An optional JSON-LD @context.

5.6.13.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Temporal Representation of an Entity whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If the *deleteAll* flag is set, remove all target Attribute instances from the target Entity.
- Otherwise:
  - if a *datasetId* parameter is provided, remove only any target Attribute instance from the given dataset;
  - otherwise remove only target Attribute instances from the target Entity that do not have a *datasetId* set.

5.6.13.5 Output data

None.

5.6.14 Modify Attribute instance in Temporal Representation of an Entity

5.6.14.1 Description

This operation allows modifying a specific Attribute (Property or Relationship) instance, identified by its *instanceId*, of a Temporal Representation of an Entity.

This operation enables the correction of wrong information that could have been previously added to the Temporal Representation of an Entity.

5.6.14.2 Use case diagram

A Context Producer can modify a specific Attribute instance, identified by a given *instanceId*, of the Temporal Representation of an Entity within an NGSI-LD system as shown in figure 5.6.14.2-1.

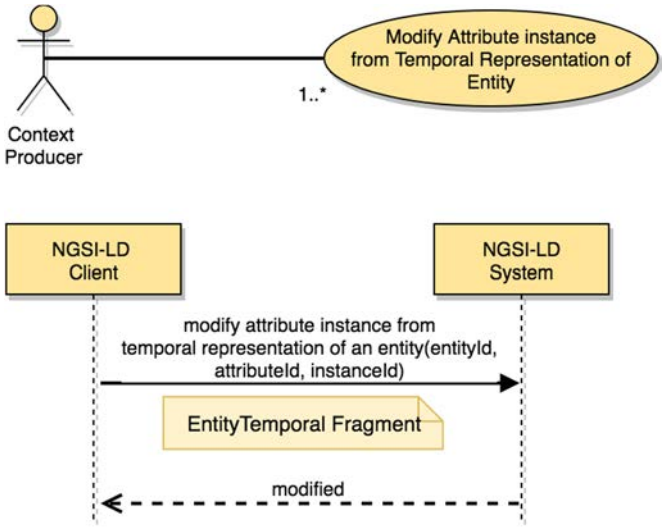

Figure 5.6.14.2-1: Modify Attribute Instance from Temporal Representation of Entity use case

5.6.14.3 Input data

- Entity id (URI) of the target Entity whose Temporal Representation is to be modified.
- Target Attribute (Property or Relationship) to be modified, identified by a Name.
- Entity Attribute instance to be modified, identified by its *instanceId*.
- A JSON-LD document representing an NGSI-LD Fragment of *EntityTemporal*, including only the new Attribute instance, contained by an Array of exactly one item.
- An optional JSON-LD @context.

5.6.14.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target *instanceId* is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If for the target Attribute no instance with the specified *instanceId* exists, an error of type *ResourceNotFound* shall be raised.
- Replace the target Attribute instance identified by the *instanceId* with the Attribute instance in the *EntityTemporal* Fragment. The *createdAt* property of the concerned instance shall remain unchanged, but the *modifiedAt* property shall be set to the timestamp corresponding to this modification.

5.6.14.5 Output data

None.

5.6.15 Delete Attribute instance from Temporal Representation of an Entity

5.6.15.1 Description

This operation allows deleting one Attribute instance (Property or Relationship), identified by its *instanceId*, of a Temporal Representation of an Entity. The Attribute itself and all its child elements shall be deleted. This operation enables the removal of individual Attribute instances that could have been previously added to the Temporal Representation of an Entity.

5.6.15.2 Use case diagram

A Context Producer can delete an Attribute instance, identified by a given *instanceId*, of the Temporal Representation of an Entity within an NGSI-LD system as shown in figure 5.6.15.2-1.

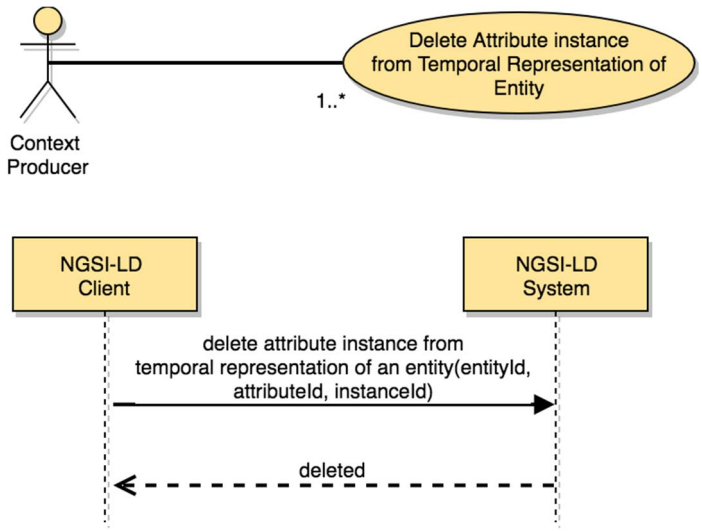

Figure 5.6.15.2-1: Delete Attribute Instance from Temporal Representation of Entity use case

5.6.15.3 Input data

- Entity id (URI) of the Entity whose Temporal Representation is to be modified, the target Entity.
- Target Attribute (Property or Relationship) to be deleted, identified by a Name.
- Entity Attribute instance to be deleted, identified by its *instanceId*.
- An optional JSON-LD @context.

5.6.15.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target *instanceId* is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.

- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the Temporal Representation of the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If for the target Attribute no instance with the specified *instanceId* exists, an error of type *ResourceNotFound* shall be raised.
- Remove the instance, with the specified *instanceId*, of the target Attribute from the target Entity.

5.6.15.5 Output data

None.

5.6.16 Delete Temporal Representation of an Entity

5.6.16.1 Description

This operation allows deleting the Temporal Representation of an Entity.

5.6.16.2 Use case diagram

A Context Producer can completely delete the Temporal Representation of an Entity within an NGSI-LD system as shown in figure 5.6.16.2-1.

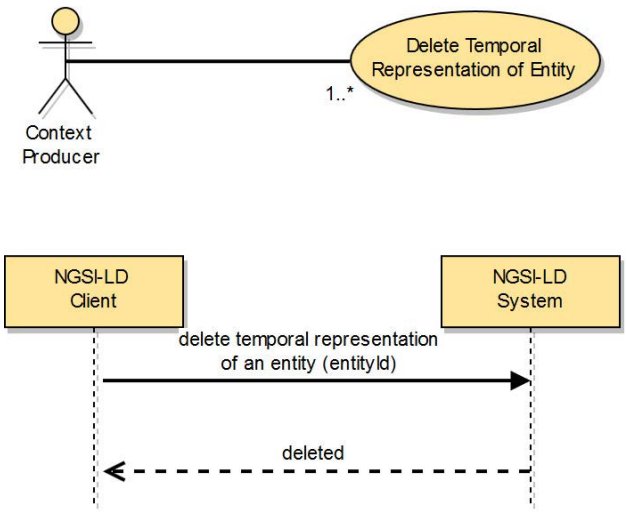

Figure 5.6.16.2-1: Delete Temporal Representation of Entity use case

5.6.16.3 Input data

- Entity Id (URI) of the target Entity, whose Temporal Representation is to be deleted.

5.6.16.4 Behaviour

- If the target Entity id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, then an error of type *ResourceNotFound* shall be raised.

- Otherwise the entire Temporal Representation of the Entity shall be removed from the Entity collection.

5.6.16.5 Output data

None.

5.7 Context Information Consumption

5.7.1 Retrieve Entity

5.7.1.1 Description

This operation allows retrieving an NGSI-LD Entity.

5.7.1.2 Use case diagram

A context consumer can retrieve a specific Entity from an NGSI-LD system as shown in figure 5.7.1.2-1.

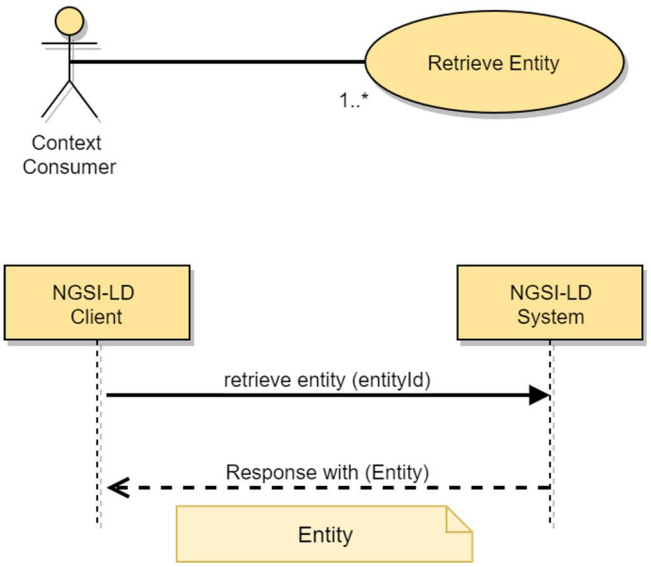

Figure 5.7.1.2-1: Retrieve Entity use case

5.7.1.3 Input data

- Entity Id (URI) of the Entity to be retrieved (target Entity).
- List of Attribute (Properties or Relationships) Names to be retrieved (projection attributes) (optional).
- An optional JSON-LD context.

5.7.1.4 Behaviour

- If the Entity Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- Otherwise return a JSON-LD object representing the Entity as mandated by clause 5.2.4 and containing only the Attributes requested (if present).

5.7.1.5 Output data

A JSON-LD object representing the target Entity as mandated by clause 5.2.4.

5.7.2 Query Entities

5.7.2.1 Description

This operation allows querying an NGSI-LD system.

5.7.2.2 Use case diagram

A context consumer can retrieve a set of entities which matches a specific query from an NGSI-LD system as shown in figure 5.7.2.2-1.

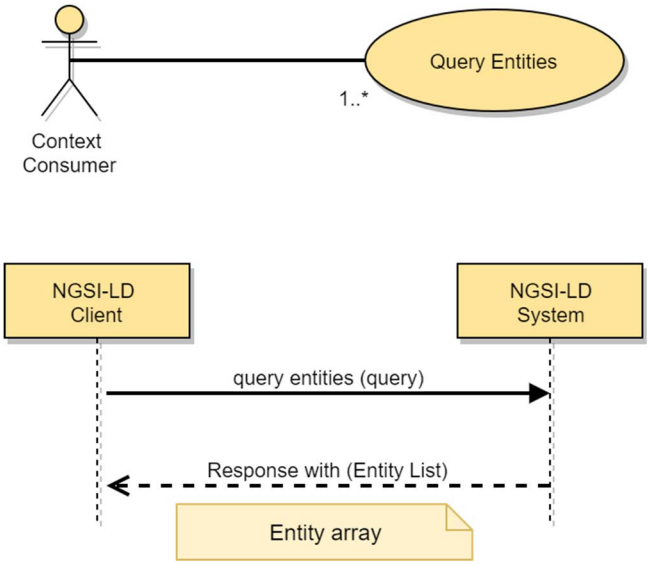

Figure 5.7.2.2-1: Query entities use case

5.7.2.3 Input data

- A reference to a JSON-LD @context (optional).
- A list (one or more) of Entity types of the matching entities (optional). Both type name (short hand string) and fully qualified type name (URI) are allowed.
- A list (one or more) of Entity identifiers (optional).
- A list (one or more) of Attribute names (query projection attributes) (optional).
- An id pattern as a regular expression (optional).
- An NGSI-LD query (values filter query) as mandated by clause 4.9 (optional).
- An NGSI-LD geo-query as mandated by clause 4.10 (optional).
- An NGSI-LD Context Source filter as per clause 4.9 (optional).
- A limit to the number of Entities to be retrieved. See clause 5.5.9.

At least one of (a) *list of Entity Types* or (b) *list of Attribute names* shall be present.

5.7.2.4 Behaviour

- If neither Entity types nor Attribute names are provided, an error of type *BadRequestData* shall be raised.
- If the list of Entity identifiers includes a URI which it is not valid, or the query or geo-query are not syntactically valid (as per the referred clauses 4.9 and 4.10) an error of type *BadRequestData* shall be raised.
- Term to URI expansion of type and Attribute names shall be observed mandated by clause 5.5.7.
- Otherwise, implementations shall run a query that shall return all the entities that meet the following conditions at the same time:
  - type matches the expanded type(s);
  - id is equivalent to any of the id(s) passed as parameter;
  - id matches the id pattern passed as parameter;
  - if present, the matching conditions specified by the query are met (as mandated by clause 4.9);
  - if present, the geospatial restrictions imposed by the geoquery are met (as mandated by clause 4.10); if there are multiple instances of the *GeoProperty* on which the geoquery is based, it is sufficient if any of these instances meets the geospatial restrictions.
- Pagination logic shall be in place as mandated by clause 5.5.9.
- If in the process of obtaining the query result it is necessary to issue a Context Source discovery operation, the same Context Source filter input parameter (if present) shall be propagated.

5.7.2.5 Output data

A JSON-LD array representing the matching entities as defined by clause 5.2.4. For each matching Entity only the Attributes specified by the Attribute list parameter shall be included. If such parameter is not present, then all Attributes shall be included.

5.7.3 Retrieve temporal evolution of an Entity

5.7.3.1 Description

This operation allows retrieving the temporal evolution of an NGSI-LD Entity.

5.7.3.2 Use case diagram

A Context Consumer can retrieve the temporal evolution of an Entity (in the form of a Temporal Representation) from an NGSI-LD system as shown in figure 5.7.3.2-1.

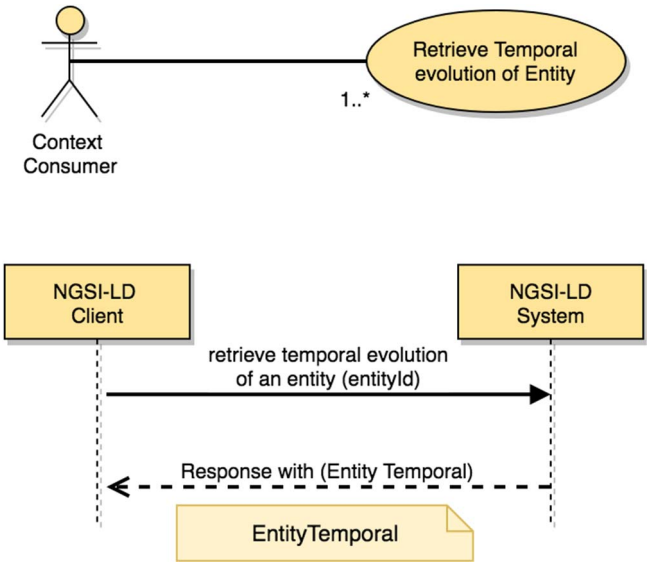

Figure 5.7.3.2-1: Retrieve temporal evolution of Entity use case

5.7.3.3 Input data

- Entity Id (URI) of the Entity, whose temporal evolution is to be retrieved (target Entity).
- List of Attribute (Properties or Relationships) Names to be retrieved (projection attributes) (optional).
- An NGSI-LD temporal query as mandated by clause 4.11 (optional).
- A parameter (*lastN*) conveying that only the last N instances (per Attribute) within the concerned temporal interval shall be retrieved (optional).
- An optional JSON-LD context.

5.7.3.4 Behaviour

- If the Entity Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- The *lastN* parameter refers to a number, *n*, of Attribute instances which shall correspond to the last *n* timestamps (in descending ordering) of the temporal property (by default *observedAt*) within the concerned temporal interval.
- Otherwise, return a JSON-LD object representing the Temporal Representation of the Entity as mandated by clause 5.2.19 and containing only the Attributes requested (if present). The NGSI-LD temporal query (if present) is used for filtering the Attribute instances. Thus, only Attribute instances, whose temporal property (explicitly specified, or *observedAt* as default) fulfils the temporal query, are included in the response, up to the number, n, specified by the *lastN* parameter (per Attribute).
    - For the avoidance of doubt, if for a requested Attribute no instance fulfils the temporal query, then an empty Array of instances shall be provided as the representation for such Attribute.

5.7.3.5 Output data

A JSON-LD object representing the Temporal Representation of the target Entity as mandated by clause 5.2.19.

5.7.4 Query temporal evolution of Entities

5.7.4.1 Description

This operation allows querying the temporal evolution of Entities present in an NGSI-LD system. It is similar to the operation defined by clause 5.7.2 (Query Entities) with the addition of a temporal query.

5.7.4.2 Use case diagram

A Context Consumer can retrieve the temporal evolution of a set of NGSI-LD Entities which matches a specific query from an NGSI-LD system as shown in figure 5.7.4.2-1.

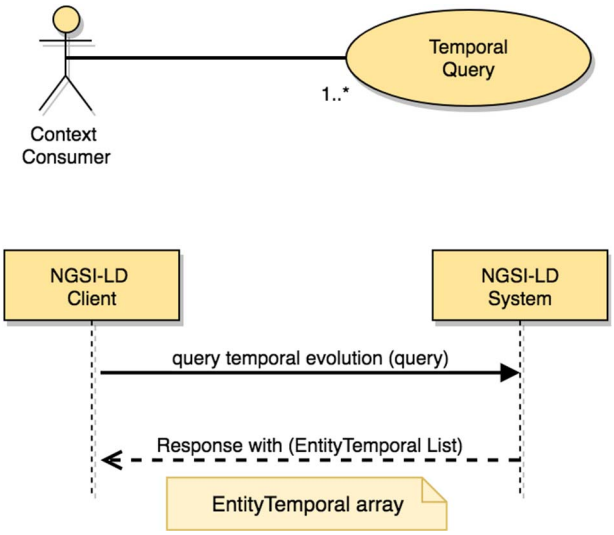

Figure 5.7.4.2-1: Temporal query use case

5.7.4.3 Input data

- A reference to a JSON-LD @context (optional).
- A list (one or more) of Attribute names (query projection attributes) (optional).
- An NGSI-LD temporal query as mandated by clause 4.11.
- A parameter (*lastN*) conveying that only the last N instances (per Attribute) within the concerned temporal interval shall be retrieved (optional).
- A list (one or more) of Entity types of the matching entities (optional). Both type name (short hand string) and fully qualified type name (URI) are allowed.
- A list (one or more) of Entity identifiers (optional).
- An id pattern as a regular expression (optional).
- An NGSI-LD query as mandated by clause 4.9 (values filter query) (optional).
- An NGSI-LD geo-query as mandated by clause 4.10 (optional).
- An NGSI-LD Context Source filter as per clause 4.9 (optional).
- A limit to the number of Entities to be retrieved. See clause 5.5.9.

At least one of (a) list of Entity Types or (b) list of Attribute names shall be present.

5.7.4.4 Behaviour

- If a temporal query is not provided then an error of type *BadRequestData* shall be raised.
- If the list of Entity identifiers includes a URI which it is not valid, or the query or geo-query are not syntactically valid (as per the referred clauses 4.9 and 4.10) an error of type *BadRequestData* shall be raised.
- Term to URI expansion of type and Attribute names shall be observed mandated by clause 5.5.7.
- The *lastN* parameter refers to a number, *n*, of Attribute instances which shall correspond to the last *n* timestamps (in descending ordering) of the temporal property (by default *observedAt*) within the concerned temporal interval.
- Otherwise, implementations shall run a query process intended to return the temporal evolution of the matching Entities; the logical steps to select the final result set of Entities, and the Attribute instances included as part of their temporal representation, are enumerated as follows:
  - Let S be the set of selected Entities i.e. the query result set.
  - If id(s) is provided, keep in S only those Entities whose id is equivalent to any of the id(s) passed as parameter.
  - If type(s) is provided, keep in S only those Entities whose Entity Type matches the expanded type(s).
  - From S, select only those Entities any of whose Attribute instances (corresponding to the Attributes specified by the query or all if none are specified) match the temporal restrictions imposed by the temporal query (as mandated by clause 4.11); i.e. if the time series, for all the concerned Attributes of an Entity, does not include data corresponding to the temporal query interval, then such Entity shall be removed from S, thus it shall not appear in the final result set. Let S1 be this new subset.
  - If a values filter query is provided, from S1, select those Entities whose Entity Attribute instances (during the interval defined by the temporal query) meet the matching conditions specified by the query (as mandated by clause 4.9); i.e. the values filter query shall be checked against all the Attribute instances resulting from the initial filtering performed by the temporal query. Let S2 be this new subset.
  - If no values filter query is provided, then S2 is equal to S1.
  - If geo-query is present, from S2, select those Entities whose *GeoProperty* instances meet the geospatial restrictions imposed by the geo-query (as mandated by clause 4.10); those geospatial restrictions shall be checked against the *GeoProperty* instances that are within the interval defined by the temporal query. Let S3 be this new subset.
  - If no geo-query is provided, then S3 is equal to S2.
  - From the set of Entities that are in S3, include in their temporal representation only the Attribute instances (up to *lastN*) corresponding to the query's projection Attributes, and which meet the temporal, query and geo-query restrictions:
    - For the avoidance of doubt, and similarly to was stated by clause 5.7.3 (Retrieve Temporal Representation of an Entity), all the query projection Attributes shall be included in the temporal representation of Entities defined by the final result set. If some of those Attributes does not include any instance for the temporal query's time interval, then it shall be represented by an empty Array.
- Pagination logic shall be in place as mandated by clause 5.5.9.
- If in the process of obtaining the query result it is necessary to issue a Context Source discovery operation, the same Context Source filter input parameter (if present) shall be propagated.

EXAMPLE: Entity Attribute: temperature

Time series values available from 2018-10-03T12:00:00 till 2018-10-03T13:00:00

Values [10,12,22,25]

Query Elements:

Temporal Query: timerel=between; time=2018-10-03T12:00:00; endTime=2018-10-03T13:00:00

Values Filter Query: q=temperature>12

As the values filter query is requesting only those temperature values which are greater than 12, even though the timeseries for the specified interval includes 4 values, i.e. 4 Attribute instances, only 2 Attribute instances (corresponding to [22,25]) will be included in the Temporal Representation of the Entity returned as part of the query result set.

5.7.4.5 Output Data

A JSON-LD array representing the matching entities as defined by clause 5.2.20 and selected according to the behavior described by clause 5.7.4.4.

5.8 Context Information Subscription

5.8.1 Create Subscription

5.8.1.1 Description

This operation allows creating a new subscription.

5.8.1.2 Use case diagram

A context subscriber can create a subscription to receive context updates within an NGSI-LD system as shown in figure 5.8.1.2-1.

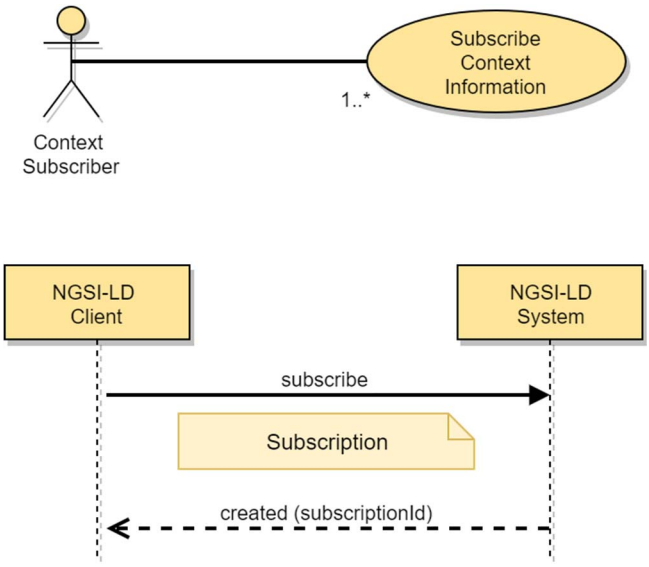

Figure 5.8.1.2-1: Create subscription use case

5.8.1.3 Input data

- A data structure (represented in JSON-LD) conforming to the *Subscription* data type as mandated by clause 5.2.12.

5.8.1.4 Behaviour

- If the data types, cardinalities and restrictions expressed by clause 5.2.12 are not met, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint already knows about this Subscription, as there is an existing Subscription whose id (URI) is equivalent, an error of type *AlreadyExists* shall be raised.
- If the subscription document does not include a Subscription identifier, a new identifier (URI) shall be automatically generated by the implementation.
- Then, implementations shall add a new Subscription to the Subscription collection. The parameters of the created Subscription shall be configured as follows:
  - The Subscription expiration date shall be equal to the value of the *expires* member. If the expiration timestamp provided represents a moment before the current date and time, then an error of type *BadRequestData* shall be raised. If there is no *expires* member the Subscription shall be considered as perpetual.
  - If the value of the *isActive* field is not included or is *true* then the initial status of the Subscription shall be set to "active".
  - If the value of the *isActive* field is *false*, then the initial status of the Subscription shall be set to "paused".
  - If present, the subscribed entities shall be those matching the conditions expressed under the *EntityInfo* collection, clause 5.2.8.
  - Watched Attributes shall be those Attributes (subject to clause 5.5.7 Term to URI expansion) pertaining to the subscribed entities (if present) and conveyed through the *watchedAttributes* member. Watched Attributes are those that trigger a new notification when they are changed. A non-present *watchedAttributes* member means that all Attributes shall be watched. If no subscribed entities have been specified, all entities with attributes matching at least one member of *watchedAttributes* are subscribed to.
- If the subscription defines a *timeInterval* member, a Notification shall be sent periodically, when the time interval (in seconds) specified in such value field is reached, regardless of Attribute changes.
- If *timeInterval* is not defined, whenever there is a change in the watched Attribute nodes (Properties or Relationships) of the concerned Entities, implementations shall post a new Notification as per the rules defined by clause 5.8.6.
- Implementations shall ensure that, when the Subscription expiration date is due, the status of the Subscription changes automatically to *expired*, so that notifications will no longer be sent.

5.8.1.5 Output data

- One subscription identifier (id) of type string, representing a URI. Implementations shall ensure that subscription identifiers are unique within an NGSI-LD system.

5.8.2 Update Subscription

5.8.2.1 Description

This operation allows updating an existing subscription.

5.8.2.2 Use case diagram

A context subscriber can update an existing subscription within an NGSI-LD system as shown in figure 5.8.2.2-1.

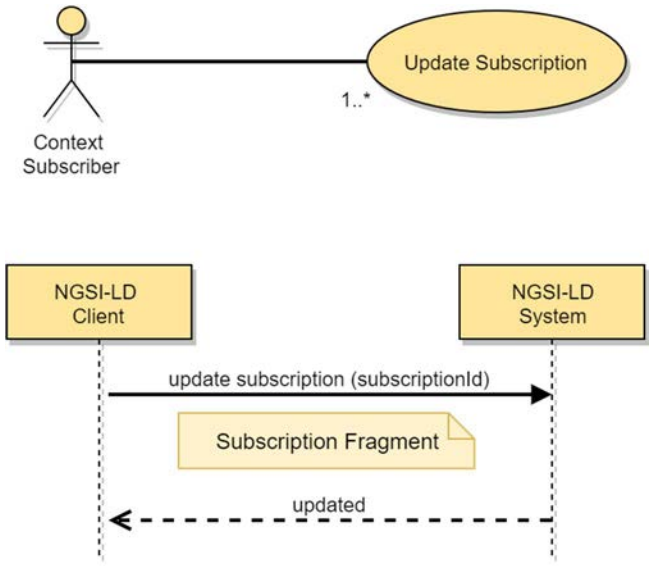

Figure 5.8.2.2-1: Update subscription use case

5.8.2.3 Input data

- Subscription identifier (URI), the target subscription.
- A JSON-LD document representing a Subscription Fragment.

5.8.2.4 Behaviour

- If the Subscription id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD System does not know about the target Subscription, because there is no existing Subscription whose id (URI) is equivalent, an error of type *ResourceNotFound* shall be raised.
- Execute the behaviour defined on clause 5.5.4 on JSON-LD validation.
- If the data types and restrictions expressed by clause 5.2.12 are not met by the *Subscription Fragment*, then an error of type *BadRequestData* shall be raised.
- Any attempt to remove (by setting them to *null* in the Fragment) mandatory properties of a Subscription (clause 5.2.12) shall result in an error of type *BadRequestData*.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- Then, implementations shall modify the target Subscription as mandated by clause 5.5.8.
- Finally, the following extra behaviour shall be observed when updating Subscriptions:
  - If *isActive* is equal to *true* or *null* and *expires* is not present, then *status* shall be updated to "active", if and only if, the previous value of *status* was different than "expired".
  - If *isActive* is equal to *true* or *null* and *expires* is *null* or corresponds to a *DateTime* in the future, then *status* shall be updated to "active".
  - If *isActive* is equal to *False* and *expires* is not present, then status shall be updated to "paused", if and only if, the previous value of *status* was different than "expired".
  - If *isActive* is *null* then *status* shall be updated to "active".
  - If only *expires* is included and refers to a *DateTime* in the future or is *null*, then *status* shall be updated to "active", if and only if the previous value of *status* was "expired".

- If *expires* is included but referring to a *DateTime* in the past, then a *BadRequestData* error shall be raised, regardless the value of *isActive*.

5.8.2.5 Output data

None.

5.8.3 Retrieve Subscription

5.8.3.1 Description

This operation allows retrieving an existing subscription.

5.8.3.2 Use case diagram

A Context Subscriber can retrieve a specific subscription from an NGSI-LD system as shown in figure 5.8.3.2-1.

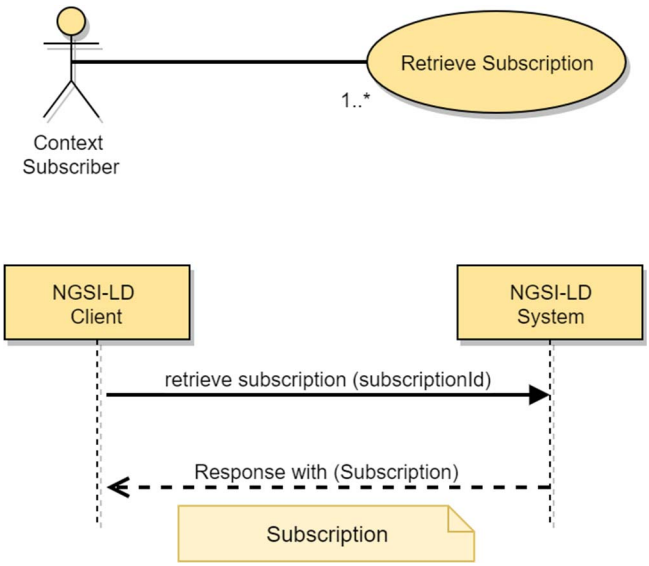

Figure 5.8.3.2-1: Retrieve subscription use case

5.8.3.3 Input data

- Id (URI) of the subscription to be retrieved (target subscription).

5.8.3.4 Behaviour

- If the subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the identifier provided does not correspond to any existing subscription in the system then an error of type *ResourceNotFound* shall be raised.
- Otherwise implementations shall query the subscription collection and obtain the subscription data to be returned to the caller.

5.8.3.5 Output data

A JSON-LD object representing the subscription details as mandated by clause 5.2.12.

5.8.4 Query Subscriptions

5.8.4.1 Description

This operation allows querying existing Subscriptions.

5.8.4.2 Use case diagram

A Context Consumer can query the existent Subscriptions from an NGSI-LD system as shown in figure 5.8.4.2-1.

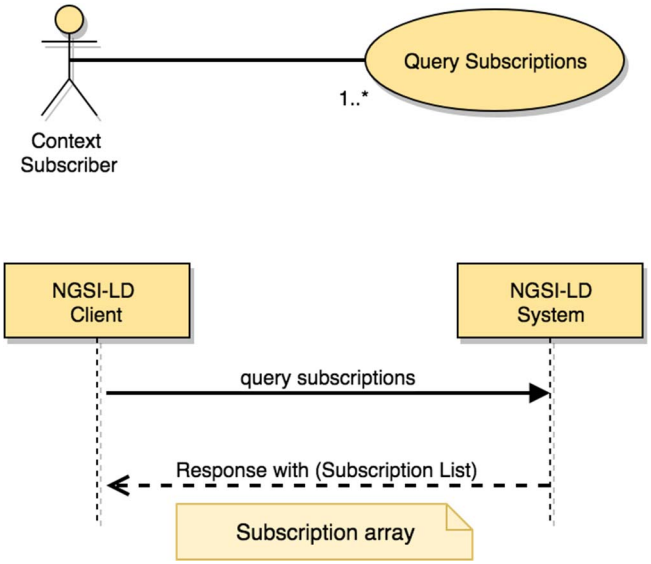

Figure 5.8.4.2-1: Query subscriptions use case

5.8.4.3 Input data

- A limit to the number of subscriptions to be retrieved. See clause 5.5.9.

5.8.4.4 Behaviour

- The NGSI-LD system shall list all the existing subscriptions up to the limit specified as input data. If no limit is specified the number of subscriptions retrieved may depend on the implementation.
- Pagination logic shall be in place as mandated by clause 5.5.9.

5.8.4.5 Output data

A list (represented as a JSON array) of JSON-LD objects each one representing subscription details as mandated by clause 5.2.12.

5.8.5 Delete Subscription

5.8.5.1 Description

This operation allows deleting an existing subscription.

5.8.5.2 Use case diagram

A context subscriber can delete a subscription within an NGSI-LD system as shown in figure 5.8.5.2-1.

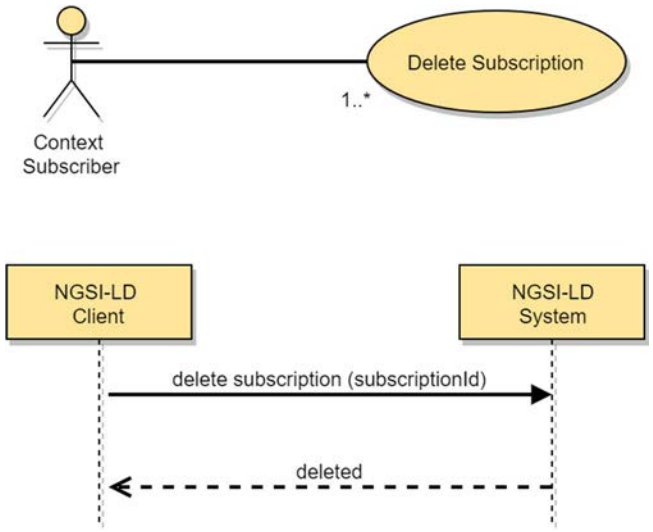

Figure 5.8.5.2-1: Delete subscription use case

5.8.5.3 Input data

- A subscription identifier (URI).

5.8.5.4 Behaviour

- If the subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the subscription id provided does not correspond to any existing subscription in the system then an error of type *ResourceNotFound* shall be raised.
- Otherwise implementations shall delete from the subscriptions collection the concerned Subscription and no longer perform notifications concerning such Subscription.

5.8.5.5 Output data

None.

5.8.6 Notification behaviour

A notification is a message that allows a subscriber to be aware of the changes in subscribed Entities. Implementations shall exhibit the following behaviour:

- Notifications shall only be sent if and only if the status of the corresponding subscription ("subscription.status") is *active*, i.e. not *paused* nor *expired*.
- If a Subscription defines a *timeInterval* member, a Notification shall be sent periodically, when the time interval (in seconds) specified in such value field is reached, regardless of Attribute changes. The notification message shall include all the subscribed Entities that match the query and geoquery conditions. If query or geoquery are not defined then all subscribed Entities shall be included.
- If a Subscription does not define a *timeInterval* term, the notification shall be sent whenever there is a change in the watched Attributes. An Attribute is considered to change when any of the members (including children) in its corresponding JSON-LD node is updated with a value different than the existing one. The notification message shall include all the subscribed Entities that changed and that match (as mandated by clauses 4.9 and 4.10) the query and geoquery conditions. If query or geoquery are not defined then all subscribed Entities that changed shall be included. If, for an Entity, there are multiple instances of the *GeoProperty* on which the geoquery is based, it is sufficient if any of these instances meets the geospatial restrictions. Finally, if a Context Source filter is defined, then only the subscribed Entities whose origin Context Source matches the referred filter shall be included.

- A Notification shall be sent as follows:
  - The structure of the notification message shall be as mandated by clause 5.3.1.
  - The Entity Attributes included (Properties or Relationships) shall be those specified by the *notification.attributes* member in the Subscription data type (clause 5.2.12). Term to URI expansion shall be observed (clause 5.5.7). The absence of the *notification.attributes* member of a Subscription means that all Entity Attributes shall be included.
  - If the *notification.format* member value is "keyValues" then a simplified representation of the entities (as mandated by clause 4.5.3) shall be provided. Otherwise the normalized format shall be used.
  - A Notification shall be sent (as mandated by each concrete binding) to the endpoint specified by the *endpoint.uri* member of the notification structure defined by clause 5.2.14. The Notification content shall be JSON by default. However, this can be changed to JSON-LD by means of the *endpoint.accept* member.
  - The *notification.timesSent* member shall be incremented by one.
  - The *notification.lastNotification* member shall be updated with a timestamp representing the current date and time.
  - If the response to the notification request is 200 OK then implementations shall:
    - Update *notification.lastSuccess* with a timestamp representing the current date and time.
    - Update *notification.status* to "ok".
  - If the response to the notification request is different than 200 OK then implementations shall:
    - Update *notification.lastFailure* with a timestamp representing the current date and time.
    - Update *notification. status* to "failed".

5.9 Context Source Registration

5.9.1 Introduction

As described in clause 5.2.9, Context Source Registrations have a similar structure as Entities and are generally handled in the same way. However, there are some aspects that are specific to Registrations, in particular with respect to the handling of required properties. Thus, the operation descriptions for Registrations reference the respective operations for Entities and in addition specify any deviations and additions that are necessary for handling Context Source Registrations.

Context Source Registrations either contain information about Context Sources providing the latest information or about Context Sources providing temporal information, but not both. Context Sources that can provide both thus have to use two separate Context Source Registrations. If no temporal query is present, only Context Source Registrations for Context Sources providing latest information are returned, i.e. those which do not specify time intervals used for temporal queries. If a temporal query is present in a request for Context Source Registrations, only those Context Source Registrations that have a matching time interval are returned.

5.9.2 Register Context Source

5.9.2.1 Description

This operation allows registering a context source within an NGSI-LD system.

5.9.2.2 Use case diagram

A context provider can register one or more context sources within an NGSI-LD system as shown in figure 5.9.2.2-1.

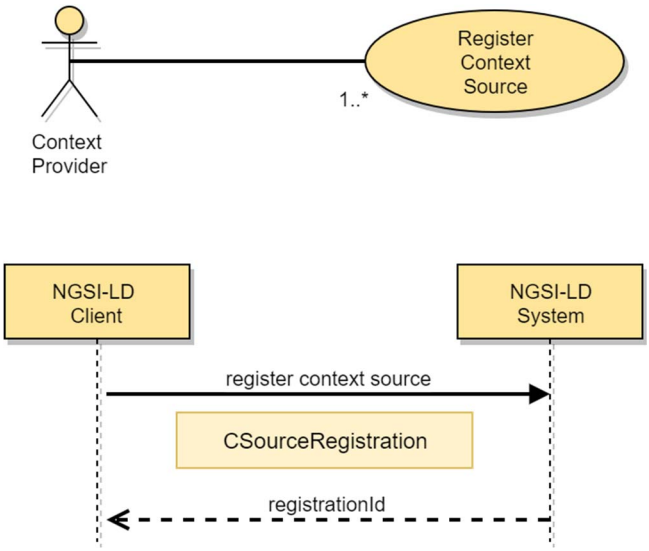

Figure 5.9.2.2-1: Register context source use case

5.9.2.3 Input data

A data structure conforming to the *CsourceRegistration* data type as mandated by clause 5.2.9.

5.9.2.4 Behaviour

Implementations shall generally exhibit the behaviour described in clause 5.6.1.4, but instead of any type of entities only Context Source Registrations can be provided. Deviating from clause 5.6.1.4, implementations shall exhibit the following behaviour:

- If the data types and restrictions expressed by clause 5.2.9 are not met by the Context Source Registration, then an error of type *BadRequestData* shall be raised.
- If the property *expires* is not defined then the Context Source Registration shall last forever (or until it is deleted from the system).
- If *expires* is a date and time in the past, an error of type *BadRequestData* shall be raised.
- If *expires* is a date and time in the future, implementations shall delete the Registration when this point in time is reached.
- If the registration identifier, *id*, is contained in the Context Source Registration, implementations have to check whether this is a valid identifier that conforms to its policies and is unique within its scope. Otherwise it can replace the 'id' with a valid registration identifier.
- Implementations shall add the concerned Context Source Registration and return an 'ok' response together with a registration identifier (*id*).
- This *id* shall be used if NGSI-LD clients need to manage the registration later.

5.9.2.5 Output data

One registration identifier (*id*) of type string, representing a URI. Implementations shall ensure that registration identifiers are unique within an NGSI-LD system.

5.9.3 Update Context Source Registration

5.9.3.1 Description

This operation allows updating a Context Source Registration in an NGSI-LD system.

5.9.3.2 Use case diagram

A Context Provider can update a Context Source Registration in an NGSI-LD system as shown in figure 5.9.3.2-1.

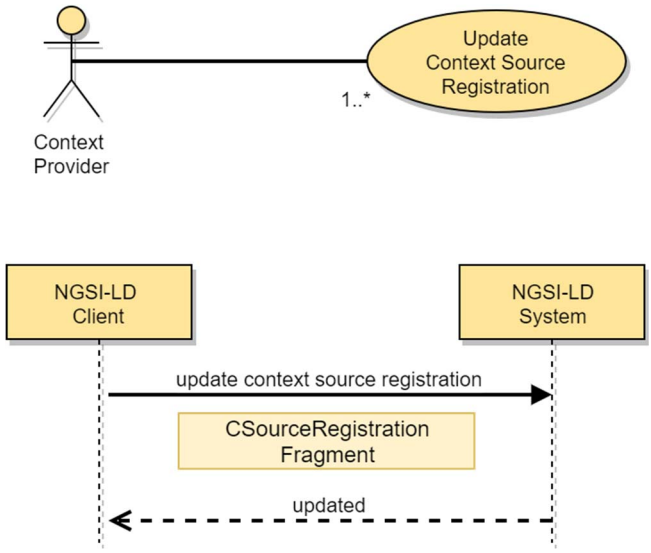

Figure 5.9.3.2-1: Update context source registration use case

5.9.3.3 Input data

- Context Source Registration identifier (URI), the target Context Source Registration.
- A JSON-LD document representing a Context Source Registration Fragment (clause 5.4).

5.9.3.4 Behaviour

- If the target Context Source Registration id (*id*) is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD System does not know about the target Context Source Registration, because there is no existing Context Source Registration whose id (URI) is equivalent, an error of type *ResourceNotFound* shall be raised.
- Execute the behaviour defined on clause 5.5.4 on JSON-LD validation.
- If the data types and restrictions expressed by clause 5.2.9 are not met by the *Context Source Registration Fragment*, then an error of type *BadRequestData* shall be raised.
- Any attempt to remove (by setting them to *null* in the Fragment) mandatory properties of a Context Source Registration (clause 5.2.9) shall result in an error of type *BadRequestData*.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- Then, implementations shall modify the target Context Source Registration as mandated by clause 5.5.8 and observing the following specific behaviour:
  - If the property *expires* is set to *null* then the Context Source Registration shall be updated to last forever.

5.9.3.5 Output data

None.

5.9.4 Delete Context Source Registration

5.9.4.1 Description

This operation allows deleting a Context Source Registration from an NGSI-LD system.

5.9.4.2 Use case diagram

A context provider can delete a context source registration from an NGSI-LD system as shown in figure 5.9.4.2-1.

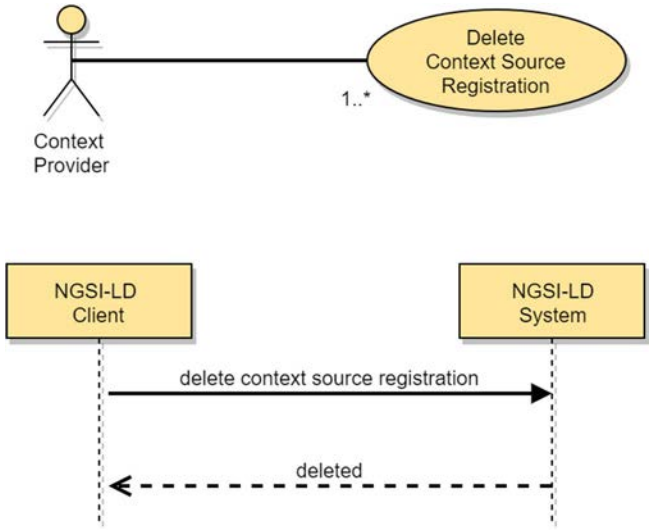

Figure 5.9.4.2-1: Delete context source registration use case

5.9.4.3 Input data

Registration identifier (URI) of the context source registration to be deleted (target registration).

5.9.4.4 Behaviour

- If the target context source registration id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target context source registration, because there is no existing context source registration whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Otherwise the context source registration shall be removed from the context source registration collection.

5.9.4.5 Output data

None.

5.10 Context Source Discovery

5.10.1 Retrieve Context Source Registration

5.10.1.1 Description

This operation allows retrieving a specific context source registration from an NGSI-LD system.

5.10.1.2 Use case diagram

A context consumer or a context provider can retrieve a specific context source registration from an NGSI-LD system as shown in figure 5.10.1.2-1.

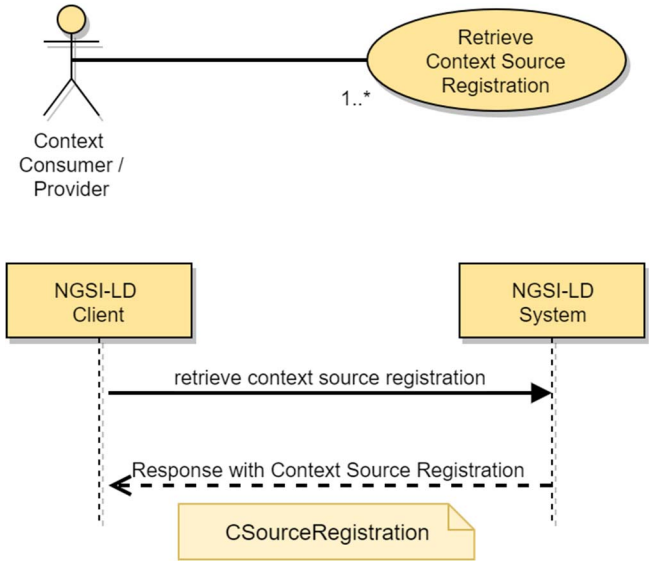

Figure 5.10.1.2-1: Retrieve context source registration use case

5.10.1.3 Input data

- Context source registration identifier (id) of the context source registration to be retrieved (target registration).

5.10.1.4 Behaviour

- If the context source registration id (*id*) is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target context source registration, because there is no existing context source registration whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- Otherwise return a JSON-LD object representing the Context Source Registration as mandated by clause 5.2.9.

5.10.1.5 Output data

A JSON-LD object representing the target context source registration as mandated by clause 5.2.9.

5.10.2 Query context source registrations

5.10.2.1 Description

This operation allows discovering context source registrations from an NGSI-LD system. The behaviour of the discovery of context source registrations differs significantly from the querying of entities as described in clause 5.7.2. The approach is that the client submits a query for entities as described in clause 5.7.2, but instead of receiving the Entity information, it receives a list of Context Source Registrations describing Context Sources that possibly have some of the requested Entity information. This means that the requested Entities and Attributes are matched against the 'information' property as described in clause 5.12.

If no temporal query is present, only Context Source Registrations for Context Sources providing latest information, i.e. without specified time intervals, are considered. If a temporal query is present only Context Source Registrations with matching time intervals, i.e. *observationInterval* or *managementInterval*, are considered.

5.10.2.2 Use case diagram

A context consumer can discover context source registrations that may be able to provide (part of) the context information specified in the query from an NGSI-LD system as shown in figure 5.10.2.2-1.

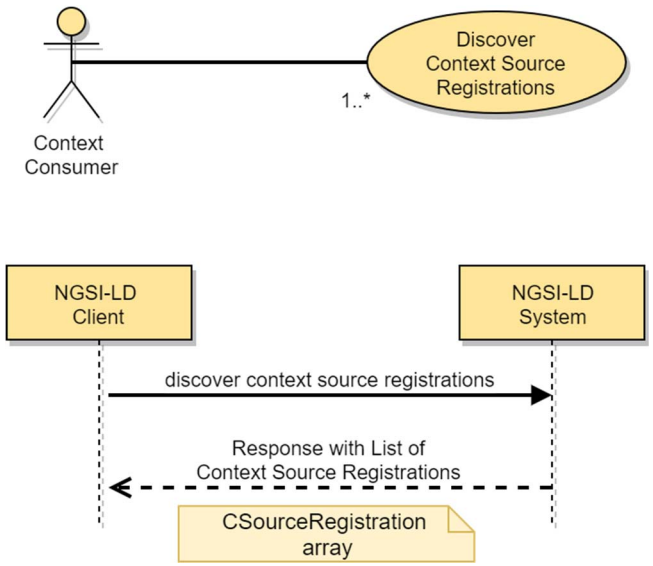

Figure 5.10.2.2-1: Discover context source registrations use case

5.10.2.3 Input data

- A reference to a JSON-LD @context (optional).
- A list (one or more) of Entity types of the matching entities (optional).
- A list (one or more) of Entity identifiers (optional).
- A list (one or more) of Attribute names (optional).
- An id pattern as a regular expression (optional).
- An NGSI-LD query (optional) as per clause 4.9.
- An NGSI-LD geo-query (optional) as per clause 4.10.
- An NGSI-LD temporal query (optional) as per clause 4.11.
- An NGSI-LD context source query (optional) as per clause 4.9.
- A limit to the number of Context Source Registrations to be retrieved. See clause 5.5.9.

At least one of (a) *list of Entity Types* or (b) *list of Attribute names* shall be present.

5.10.2.4 Behaviour

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- If neither Entity types nor Attribute names are provided, an error of type *BadRequestData* shall be raised.
- If the list of Entity identifiers includes a URI which it is not valid, or the query, geo-query or temporal query are not syntactically valid (as per clauses 4.9, 4.10 and 4.11) an error of type *BadRequestData* shall be raised.

- If a JSON-LD context is not provided then all the query terms shall be resolved against the default JSON-LD @context.
- Implementations should run a query that shall return context source registrations that meet all the applicable conditions:
  - If present, the entity specification in the query consisting of a combination of entity type and entity id/entity id pattern matches an *EntityInfo* specified in a *RegistrationInfo* of the information property in a context source registration. If there is no *EntityInfo* specified in the *RegistrationInfo*, the entity specification is considered matching. This matching is further described in clause 5.12.
  - If present, at least one Attribute name specified in the query matches one Property or Relationship in the *RegistrationInfo* element of the information property in a context source registration.. If no Properties or Relationships are specified in the *RegistrationInfo*, the Attribute names are considered matching. This matching is further described in clause 5.12.
  - If present, the geoquery is matched against the *GeoProperty* identified in the geoquery. If no *GeoProperty* is specified in the geoquery, the default property is 'location'. The geoquery matches the *GeoProperty* specified in the Context Source Registration, if the location directly matches or if the location possibly contains locations that would match the geoquery.
  - If no temporal query is present, only Context Source Registrations for Context Sources providing latest information, i.e. without specified time intervals, are considered.
  - If a temporal query is present, only Context Source Registrations with specified time intervals, i.e. *observationInterval* or *managementInterval* are considered. If the *timeproperty* is *observedAt* or no *timeproperty* is specified in the temporal query (default: *observedAt*), the temporal query is matched against the *observationInterval* (if present). If the *timeproperty* is *createdAt* or *modifiedAt*, the temporal query is matched against the *managementInterval* (if present). If the relevant interval is not present, there is no match:
    - The semantics of the match is that the "time" in the case of the "before" and "after" relation is contained in or is an endpoint of a time period included in the specified time interval. In the case of the "between" relation there is a match if there is an overlap between the interval specified by the "time" and "endtime" and the specified time interval.
  - If present, the conditions specified by the context source query match the respective Context Source Properties (as mandated by clause 4.9).
- Pagination logic shall be in place as mandated by clause 5.5.9.

5.10.2.5 Output data

A JSON-LD array of matching Context Source Registrations as defined by clause 5.2.9. Instead of the original Context Source Registration which may contain a lot of irrelevant information, implementations should return filtered Context Source Registrations, which only contain context source registration information relevant for the query, in particular only matching *RegistrationInfo* elements.

5.11 Context Source Registration Subscription

5.11.1 Introduction

Context Source Registration Subscriptions in general work like context information subscriptions; however, instead of resulting in notifications with context information, the notifications contain Context Source Registrations describing Context Sources that can potentially provide the requested context information. If no temporal query is present, only Context Source Registrations for Context Sources providing latest information, i.e. without such time intervals, are considered. If a temporal query is present only Context Source Registrations with matching time intervals, i.e. *observationInterval* or *managementInterval*, are considered.

5.11.2 Create Context Source Registration Subscription

5.11.2.1 Description

This operation allows creating a new Context Source Registration Subscription.

5.11.2.2 Use case diagram

A Context Source subscriber can subscribe to a new Context Source Registration Subscription as shown in figure 5.11.2.2-1.

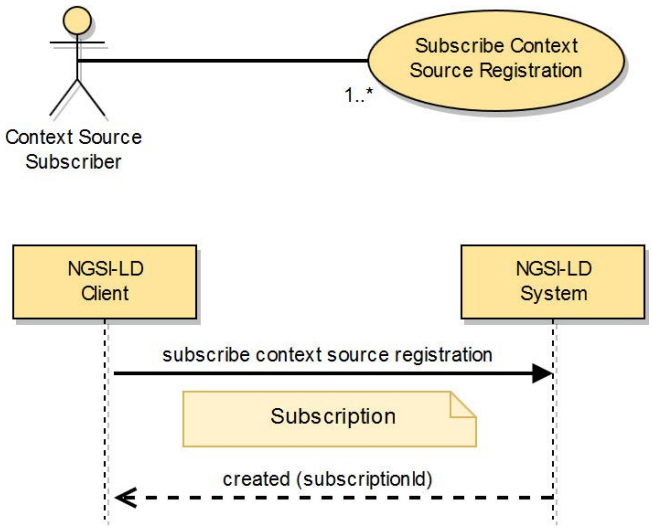

Figure 5.11.2.2-1: Subscribe Context Source Registration use case

5.11.2.3 Input data

- A data structure (represented in JSON-LD) conforming to the Subscription data type as mandated by clause 5.2.12.

5.11.2.4 Behaviour

- The behaviour shall be as described in clause 5.8.1.4 with the following exceptions:
  - If all checks described in clause 5.8.1.4 are passed, implementations shall add a new subscription to the Context Source Registration Subscription collection. The parameters of the created subscription shall be configured as described in clause 5.8.1.4.
  - Instead of directly matching the entities and watched Attributes from the subscription with the Context Source registrations, the entities specified in the subscription, the watched Attributes and the Attributes specified in the notification parameter are matched against the respective *information* property of the Context Source registrations. If either the watched Attributes or the Attributes in the notification are not present or of length 0, all possible Attributes (if present in the Context Source Registrations) for matching entities match. This matching is further described in clause 5.12.
  - If present, the geoquery in the geoQ element is matched against the *GeoProperty* of the subscription identified in the geoQ element. If no *GeoProperty* is specified in the geoquery, the default property is 'location'. The geoquery matches the *GeoProperty* specified in the Context Source Registration, if the location directly matches or if the location possibly contains locations that would match the geoquery.
  - If no temporal query is present in the *temporalQ* element, only Context Source Registrations for Context Sources providing latest information, i.e. without specified time intervals for *observationInterval* or *managementInterval*, are considered.

- If a temporal query in the *temporalQ* element is present, only Context Source Registrations with specified time intervals are considered. If the *timeproperty* is *observedAt* or no *timeproperty* is specified in the temporal query (default: *observedAt*), the temporal query is matched against the *observationInterval* (if present). If the *timeproperty* is *createdAt* or *modifiedAt*, the temporal query is matched against the *managementInterval* (if present). If the relevant interval is not present, there is no match:
  - The semantics of the match is that the "time" in the case of the "before" and "after" relation is contained in or is an endpoint of a time period included in the specified time interval. In the case of the "between" relation there is a match if there is an overlap between the interval specified by the "time" and "endtime" and the specified time interval.

- If the subscription defines a "timeInterval" term, a *cSourceNotification* (clause 5.3.2) with all matching Context Source Registrations shall be sent periodically, initially on subscription and when the time interval (in seconds) specified in such value field is reached, independent of any changes to the set of Context Source registrations.
- If "timeInterval" is not defined, initially on subscription and whenever there is a change of a matching Context Source Registration (creation, update, deletion), implementations shall post a new *cSourceNotification* to the endpoint specified in the notification parameters informing about this change by providing the Context Source Registration(s) together with the appropriate trigger reason in the "triggerReason" member.

5.11.2.5 Output data

One subscription identifier (id) of type string, representing a URI. Implementations shall ensure that subscription identifiers are unique within an NGSI-LD system.

5.11.3 Update Context Source Registration Subscription

5.11.3.1 Description

This operation allows updating an existing Context Source Registration Subscription.

5.11.3.2 Use case diagram

A context source subscriber can update a Context Source Registration Subscription. as shown in figure 5.11.3.2-1.

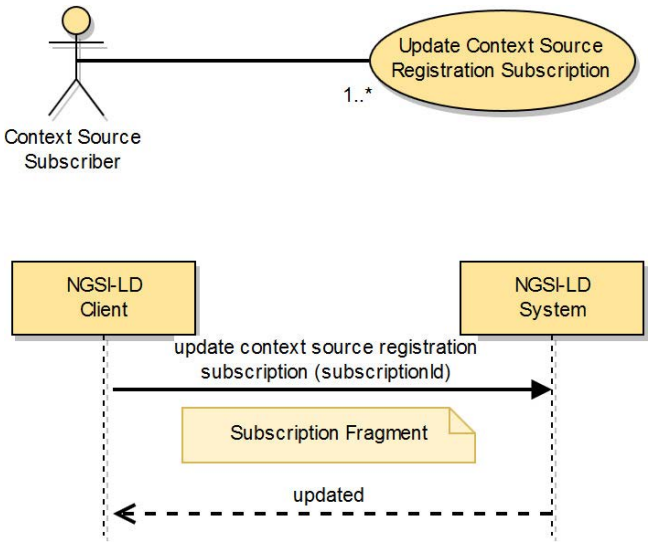

Figure 5.11.3.2-1: Update Context Source Registration Subscription use case

5.11.3.3 Input data

- Subscription identifier (URI), the target Context Source Registration Subscription.
- A JSON-LD document representing a Subscription Fragment.

5.11.3.4 Behaviour

- If the Subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the data types and restrictions expressed by clause 5.2.12 are not met by the Subscription Fragment, then an error of type *BadRequestData* shall be raised.
- Any attempt to remove (by setting them to *null* in the Fragment) mandatory properties of a Context Source Registration Subscription (clause 5.2.9) shall result in an error of type *BadRequestData*.
- Then, implementations shall modify the target subscription as mandated by clause 5.5.8.
- Finally, send a notification with all currently matching Context Source Registrations.

5.11.3.5 Output data

None.

5.11.4 Retrieve Context Source Registration Subscription

5.11.4.1 Description

This operation allows retrieving an existing Context Source Registration Subscription.

5.11.4.2 Use case diagram

A Context Source subscriber can retrieve a specific Context Source Registration Subscription as shown in figure 5.11.4.2-1.

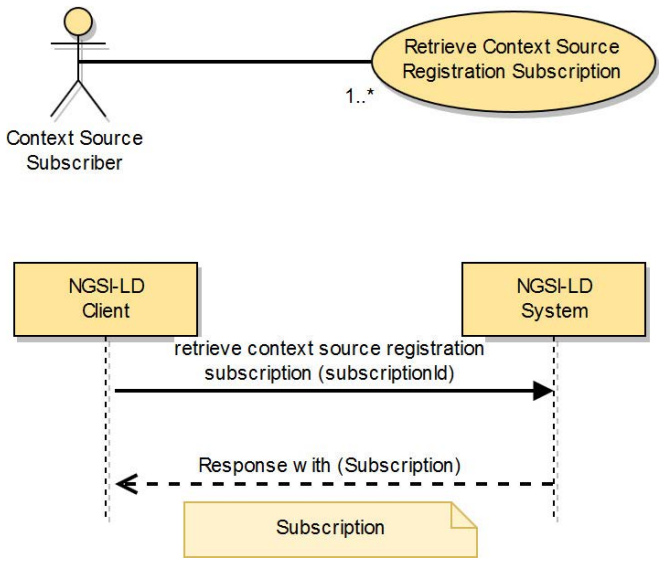

Figure 5.11.4.2-1: Retrieve Context Source Registration Subscription use case

5.11.4.3 Input data

- Id (URI) of the subscription to be retrieved (target subscription).

5.11.4.4 Behaviour

- If the subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the identifier provided does not correspond to any existing subscription in the system then an error of type *ResourceNotFound* shall be raised.
- Otherwise implementations shall query the Context Source Registration Subscription collection and obtain the subscription data to be returned to the caller.

5.11.4.5 Output data

A JSON-LD object representing the subscription details as mandated by clause 5.2.12.

5.11.5 Query Context Source Registration Subscriptions

5.11.5.1 Description

This operation allows querying existing Context Source Registration Subscriptions.

5.11.5.2 Use case diagram

A context source subscriber can query all existing Context Source Registration Subscriptions as shown in figure 5.11.5.2-1.

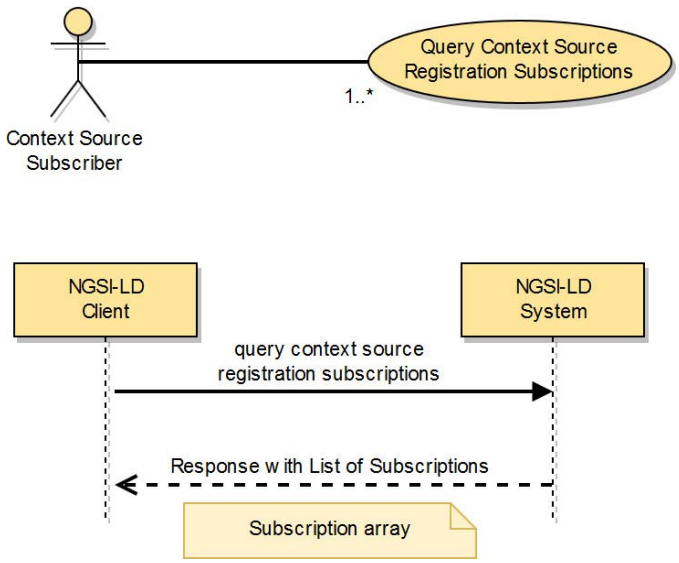

Figure 5.11.5.2-1: Retrieve Context Source Registration Subscriptions use case

5.11.5.3 Input data

- A limit to the number of Context Source Registration Subscriptions to be retrieved. See clause 5.5.9.

5.11.5.4 Behaviour

- The NGSI-LD System shall list all the existing Context Source Registration Subscriptions.
- Pagination logic shall be in place as mandated by clause 5.5.9.

5.11.5.5 Output data

A list (represented as a JSON array) of JSON-LD objects each one representing subscription details as mandated by clause 5.2.12.

5.11.6 Delete Context Source Registration Subscriptions

5.11.6.1 Description

This operation allows deleting an existing Context Source Registration Subscription.

5.11.6.2 Use case diagram

A context source subscriber can delete a Context Source Registration Subscription as shown in figure 5.11.6.2-1.

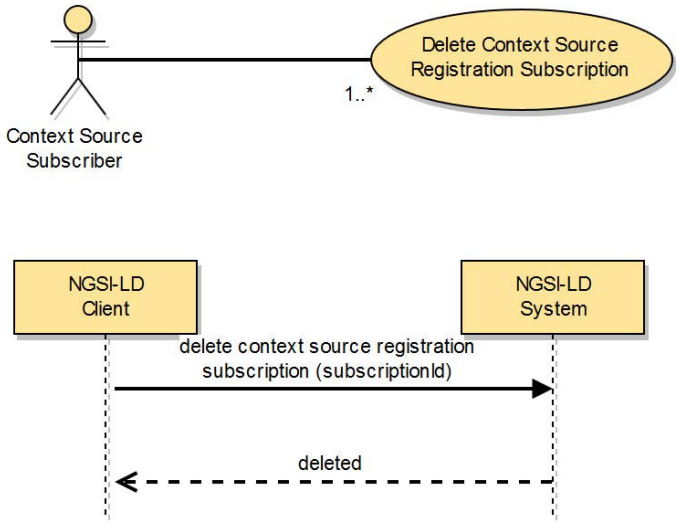

Figure 5.11.6.2-1: Delete Context Source Registration Subscriptions use case

5.11.6.3 Input data

- A subscription identifier (URI).

5.11.6.4 Behaviour

- If the subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the subscription id provided does not correspond to any existing subscription in the system then an error of type *ResourceNotFound* shall be raised.
- Otherwise implementations shall delete from the Context Source Registration Subscriptions collection the concerned subscription and no longer perform notifications concerning such subscription.

5.11.6.5 Output data

None.

5.11.7 Notification behaviour

A Context Source Notification is a message that allows a subscriber to be aware of the changes in the set of Context Source Registrations describing Context Sources that can potentially provide the requested context information. Implementations shall exhibit the behaviour described in clause 5.8.6 with the following exceptions:

- If a subscription defines a "timeInterval" member, a *CsourceNotification* (clause 5.3.2) shall be sent on initial subscription and periodically, when the time specified time interval (in seconds) has elapsed, regardless of any changes to the set of context source registrations. The *CsourceNotification* message shall include all the Context Source Registrations whose *information* property matches the entities and watched Attributes or Attributes specified in the notification parameter and, if present, have a matching geo-query. If either the watched Attributes or the Attributes in the notification are not present or of length 0, all possible Attributes (if present in the Context Source Registrations) for fitting entities match.

- If a subscription does not define a "timeInterval" term, the csource notification shall be sent on initial subscription and whenever there is a change in a matching csource registration. Such a change may be triggered by the creation of a new matching csource registration, the update of a csource registration (whether matching before the update, after the update or in both cases) or the deletion of a matching csource registration. The notification message shall include the matching csource registration(s) together with the appropriate trigger reason in the "triggerReason" member.

- Instead of providing the original Context Source Registration which may contain a lot of irrelevant information, implementations should return filtered Context Source Registrations, which only contain context source registration information relevant for the subscription, in particular only matching *RegistrationInfo* elements.

- A csource notification shall be sent as follows:

  - The structure of the csource notification message shall be as mandated by clause 5.3.2.

  - A csource notification shall be sent to the "endpoint".

  - The "notification.timesSent" member shall be incremented by one.

  - The "notification.lastNotification" member shall be updated with the current timestamp.

  - If the notification is sent successfully:

    - Update "notification.lastSuccess" with the current timestamp.

  - If the notification is not sent successfully:

    - Update "notification.lastFailure" with the current timestamp.

    - Update the subscription "status" to "failed".

5.12 Matching Context Source Registrations

When querying Context Source Registrations as described in clause 5.10.2 and subscribing to Context Source Registrations as described in clause 5.11.2, the Entities and/or Attributes specified in the request have to be matched against the set of Context Source Registrations, extracting the matching ones. This clause describes this matching.

The relevant specification information in the query for Context Source Registrations are the list of Entity Types (if present), the list of Entity identifiers (if present), the id pattern (if present) and the list of Attribute names (if present). In the case of subscriptions to context source registrations, it is the Entities as specified in the array of type *EntityInfo* in the Subscription, the *watchedAttributes* element of the *Subscription* and the attributes specified as part of the *NotificationParams* element of the Subscription. If the attributes in the *NotificationParams* element are empty or not present, the matching is done as if no attribute identifiers have been specified, otherwise the combination of the *watchedAttributes* and the attributes in the *NotificationParams* element are used as the specified attribute identifiers for the matching.

Even though the structure of Entity specifications differs in queries and subscriptions, they consist of the same information, so for the purpose of this clause, the Entity specification refers to the relevant elements for matching, i.e. Entity Types, Entity identifiers, id pattern and Attribute names. An Entity specification shall contain at least one of:

a) list of Entity Types; or

b) list of Attribute names.

An Entity specification matches a Context Source Registration if at least one of the *RegistrationInfo* elements in the *information* element matches. An Entity specification matches a *RegistrationInfo* if the following conditions hold:

- If present, the Entity Types, Entity identifiers and id pattern match at least one of the *EntityInfo* elements (see below).
- If present, the Attribute identifiers match the combination of Properties and Relationships specified in the *RegistrationInfo* (see below).

An Entity specification consisting of Entity Types, Entity identifiers and id pattern matches an *EntityInfo* element if one of the specified Entity Types matches the entity type in the *EntityInfo* element and one of the following conditions holds:

- The *EntityInfo* contains neither an *id* nor an *idPattern*.
- One of the specified entity identifiers matches the *id* in the *EntityInfo*.
- At least one of the specified entity identifiers matches the *idPattern* in the *EntityInfo*.
- The specified id pattern matches the *id* in the *EntityInfo*.
- Both a specified id pattern and an *idPattern* in the *Entity Info* are present (since in the general case it is not easily feasible to determine if there can be identifiers matching both patterns).

Attribute names match the combination of Properties and Relationships if one of the following conditions hold:

- No Attribute names have been specified (as this means all Attributes are requested).
- The combination of Properties and Relationships is empty (as this means only Entities have been registered and the Context Sources may have matching Property or Relationship instances).
- If at least one of the specified attribute names matches a Property or Relationship specified in the *RegistrationInfo*.

6 API HTTP binding

6.1 Introduction

This clause defines the resources and operations of the NGSI-LD API. The NGSI-LD API is structured in terms of HTTP [3], [4] verbs, request and response payload bodies.

A non-normative OAS specification [i.12] of the referred HTTP binding can be found at [i.14].

6.2 Global definitions and resource structure

All resource URIs of this API shall have the following root:

- {apiRoot}/{apiName}/{apiVersion}/

NOTE 1: The *apiRoot* discovery process is out of the scope of the present document.

NOTE 2: The *apiRoot* for Context Source related aspects and the *apiRoot* for general Entity-related aspects can be different, e.g. the Context Source related aspects can be implemented by a Context Registry as shown for the distributed and federated architectures (see clause 4.3), whereas the Entity-related aspects would be implemented by a Context Broker.

NOTE 3: The *apiRoot* for Context Source related aspects and the *apiRoot* for general Entity-related aspects can be different than the *apiRoot* for temporal aspects, e.g. the temporal aspects can be implemented by an NGSI-LD subsystem specialized in historical data.

The *apiRoot* includes the scheme ("http" or "https"), host and optional port, and an optional prefix string. The API shall support HTTP over TLS (also known as HTTPS - see IETF RFC 2818 [18]). TLS version 1.2 as defined by IETF RFC 5246 [19] shall be supported. HTTP without TLS is not recommended.

The *apiName* shall be set to "ngsi-ld" and the *apiVersion* shall be set to "v1" for the present document.

All resource URIs in clauses 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11, 6.12, 6.13, 6.14, 6.15, 6.16, 6.17, 6.18, 6.19, 6.20, 6.21, 6.22 are defined relative to the above root URI. The structure of the resources under the root URI is shown in figure 6.2-1 and methods defined on them are shown in table 6.2-1.

Figure 6.2-1: Resource URI structure of the NGSI-LD API

Table 6.2-1: Resources and HTTP methods defined on them

Resource Name	Resource URI	HTTP Method	Meaning
Entity List	/entities/	POST	Entity creation
		GET	Query entities
Entity by id	/entities/{entityId}	GET	Entity retrieval by id
		DELETE	Entity deletion by id
Entity Attribute List	/entities/{entityId}/attrs/	POST	Append entity Attributes
		PATCH	Update entity Attributes
Attribute by id	/entities/{entityId}/attrs/{attrId}	PATCH	Attribute partial update
		DELETE	Attribute delete
Subscriptions List	/subscriptions/	POST	Subscription creation
		GET	Subscription list retrieval
Subscription by Id	/subscriptions/{subscriptionId}	GET	Subscription retrieval by id
		PATCH	Subscription update by id
		DELETE	Subscription deletion by id
Context source registration list	/csourceRegistrations/	POST	Csource registration creation
		GET	Discover Csource registrations
Context source registration by Id	/csourceRegistrations/{registrationId}	GET	Csource registration retrieval by id
		PATCH	Csource registration update by id
		DELETE	Csource registration deletion by id
Context source registration subscription list	/csourceSubscriptions/	POST	Csource registration subscription
		GET	Csource registration subscription list retrieval
Context source registration subscription by Id	/csourceSubscriptions/{subscriptionId}	GET	Csource registration subscription retrieval by id
		PATCH	Csource registration subscription update by id
		DELETE	Csource registration subscription deletion by id
Entity Operations. Create	/entityOperations/create	POST	Batch Entity creation
Entity Operations. *Upsert*	/entityOperations/upsert	POST	Batch Entity create or update (*upsert*)
Entity Operations. Update	/entityOperations/update	POST	Batch Entity update
Entity Operations. Delete	/entityOperations/delete	POST	Batch Entity deletion
Entity Temporal Evolution	/temporal/entities/	POST	Temporal Representation of Entity creation
		GET	Query temporal evolution of Entities
Temporal Representation of Entity by id	/temporal/entities/{entityId}	GET	Temporal Representation of Entity retrieval by id
		DELETE	Temporal Representation of Entity deletion by id
Temporal Representation of Entity Attribute List	/temporal/entities/{entityId}/attrs/	POST	Temporal Representation of Entity Attribute instance addition
Temporal Representation of Entity Attribute by id	/temporal/entities/{entityId}/attrs/{attrId}	DELETE	Attribute from Temporal Representation of Entity deletion
Temporal Representation of Entity Attribute Instance by id	/temporal/entities/{entityId}/attrs/{attrId} /{instanceId}	PATCH	Attribute Instance update
		DELETE	Attribute Instance deletion by instance id

6.3 Common behaviours

6.3.1 Introduction

This clause extends the API common behaviours to the particularities of the HTTP REST binding. For each operation implementations shall exhibit the common behaviours as specified by clause 5.5 and the behaviours defined by the present clause.

6.3.2 Error types

This clause associates API error types (which shall be contained in the response payload body) defined by clause 5.5.2 with HTTP status codes as shown in table 6.3.2-1.

Table 6.3.2-1: Mapping of error types to HTTP status codes

Error Type	HTTP status
https://uri.etsi.org/ngsi-ld/errors/InvalidRequest	400
https://uri.etsi.org/ngsi-ld/errors/BadRequestData	400
https://uri.etsi.org/ngsi-ld/errors/AlreadyExists	409
https://uri.etsi.org/ngsi-ld/errors/OperationNotSupported	422
https://uri.etsi.org/ngsi-ld/errors/ResourceNotFound	404
https://uri.etsi.org/ngsi-ld/errors/InternalError	500
https://uri.etsi.org/ngsi-ld/errors/TooComplexQuery	403
https://uri.etsi.org/ngsi-ld/errors/TooManyResults	403
https://uri.etsi.org/ngsi-ld/errors/LdContextNotAvailable	503

In addition, implementations shall support the standard specific errors of HTTP bindings, such as the following:

- "Method Not Allowed" (405) which shall be raised when a client invokes a wrong HTTP verb over a resource. Implementations shall provide the allowed HTTP methods as mandated by IETF RFC 7231 [3] in section 6.5.5.
- "Request Entity too large" (413) which shall be raised when the HTTP input data stream provided by a client was too large i.e. too many bytes.
- "Length required" (411) which shall be raised when an HTTP request provided by a client does not define the "Content-Length" HTTP header.
- "Unsupported Media Type" (415) which shall be raised when an HTTP request payload body (as per the "Content-Type" header) it is not "application/json" nor "application/ld+json".
- "Not Acceptable" (406) which shall be raised when the response media types that are acceptable by a client (as per the "Accept" header) do not include or expand to "application/json" nor "application/ld+json".

6.3.3 Reporting errors

When an API operation results in an error, implementations shall return an HTTP response as follows:

- Content-Type: application/json.
- HTTP Status Code: As per clause 6.3.2 depending on error type.
- Payload body: A JSON object including all the terms defined by clause 5.5.3.

6.3.4 HTTP request preconditions

For POST and PATCH HTTP requests implementations shall check the following preconditions:

- Content-Type header shall be "application/json" or "application/ld+json".
- Content-Length header shall include the length of the request payload body.

For PATCH HTTP requests "application/merge-patch+json" is allowed as Content-Type, as mandated by IETF RFC 7396 [16]. Implementations shall interpret such MIME type as equivalent to "application/json".

For GET HTTP requests implementations shall check the following preconditions:

- Accept header shall include (or define a media range that can be expanded to) "application/json" or "application/ld+json".

When Accept header is present and can be expanded to both "application/json" and "application/ld+json", "application/ld+json" shall be preferred, unless the HTTP Accept header processing rules, e.g. the presence of a "q" parameter indicating a relative weight, (as mandated by IETF RFC 7231 [3], section 5.3.2) require otherwise. If the Accept header is not present "application/json" shall be assumed.

If an incoming HTTP request does not meet the preconditions stated above, an HTTP error response of type *InvalidRequest* shall be returned, with the following exceptions:

- "Content-Length" HTTP header absence, shall result in just a **411** HTTP status code (without any payload body).
- Unsupported Media Type, i.e. "Content-Type" header is not "application/json" nor "application/ld+json", shall result in just a **415** HTTP status code (without any payload body).
- Not Acceptable Media Type, i.e. "Accept" header does not imply "application/json" nor "application/ld+json", shall result in just a **406** HTTP status code (without any payload body).

6.3.5 JSON-LD @context resolution

In the HTTP REST binding, implementations shall resolve the JSON-LD "@context" associated to an incoming HTTP request as follows:

- If the request verb is GET or DELETE, then the associated JSON-LD "@context" shall be obtained from a Link header [7] as mandated by JSON-LD [2], clause 6.8. In the absence of such Link header, then the associated "@context" shall be the default JSON-LD "@context".

  EXAMPLE: The structure of the referred Link header is shown below. The first component (between < >) is a dereferenceable URI pointing to the JSON-LD document which contains the @context to be used to expand the terms used by the corresponding operation. The second parameter is a fixed, non-dereferenceable URI used to denote a unique identifier and semantics for this header (marking it as a link to a JSON-LD @context). The third and final parameter flags the MIME type of the linked resource (JSON-LD).

  Link: <http://json-ld.org/contexts/person.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json".

- If the request verb is POST or PATCH and the Content-Type header is "application/json", then the @context shall be obtained from a Link header as mandated by JSON-LD [2], clause 6.8. In the absence of such Link header, then the "@context" shall be the default @context. In any case, if the request payload body (as JSON) contains a "@context" term, then an HTTP error response of type *BadRequestData* shall be raised.
- If the request verb is POST or PATCH and the Content-Type header is "application/ld+json", then the associated @context shall be obtained from the request payload body itself. If no @context can be obtained from the request payload body, then an HTTP error response of type *BadRequestData* shall be raised. In any case, the presence of a JSON-LD Link header in the incoming HTTP request when the Content-Type header is "application/ld+json" shall result in an HTTP error response of type *BadRequestData*.

In summary, from a developer's perspective, for POST and PATCH operations, if MIME type is "application/ld+json", then the associated @context shall be provided only as part of the request payload body. Likewise, if MIME type is "application/json", then the associated @context shall be provided only by using the JSON-LD Link header. No mixes are allowed, i.e. mixing options shall result in HTTP response errors. Implementations should provide descriptive error messages when these situations arise.

On the other hand, GET and DELETE operations always take their input @context from the JSON-LD Link header.

6.3.6 HTTP response common requirements

Implementations shall honour the Accept header provided by HTTP requests as mandated by clause 6.3.4:

- If the target response's MIME type is "application/json" such response shall include a Link to the associated JSON-LD @context as mandated by [2], clause 6.8.
- If the target response's MIME type is "application/ld+json", then the response payload body provided by the HTTP response shall include a JSON-LD @context.

For operations whereby the response payload body is not present, such as POST or PATCH operations (responding with 201 and 204 HTTP response codes), implementations shall provide in their HTTP response a JSON-LD Link header referencing the relevant JSON-LD @context that could be used later, regardless of the value of the Accept header.

6.3.7 Simplified representation of entities

For HTTP GET operations performed over the resource /entities and all of its sub-resources, implementations shall support the parameter specified in table 6.3.7-1.

Table 6.3.7-1: Simplified representation: options parameter

Name	Data type	Cardinality	Remarks
options	Comma separated list of strings	0..1	When its value includes the keyword "keyValues", a simplified representation of entities shall be provided as defined by clause 4.5.3.

6.3.8 Notification behaviour

In the HTTP binding a notification shall be sent by issuing an HTTP POST request targeted to the value of "endpoint.uri" member of the subscription structure defined by clause 5.2.12. The MIME type associated to the POST request shall be "application/json" by default. However, this can be changed to application/ld+json by means of the "endpoint.accept" member.

If the target MIME type is "application/json" then the HTTP notification request shall include a Link header with a reference to the corresponding JSON-LD @context as mandated by the JSON-LD specification [2], clause 6.8 (to the default JSON-LD @context if none available).

6.3.9 Csource Notification behaviour

In the HTTP binding a csource notification shall be sent by issuing an HTTP POST request targeted to the value of "endpoint.uri" member of the csource subscription structure defined by clause 5.2.12. The MIME type associated to the POST request shall be "application/json" by default. However, this can be changed to application/ld+json by means of the "endpoint.accept" member.

If the target MIME type is "application/json" then the HTTP notification request shall include a Link header with a reference to the corresponding JSON-LD @context as mandated by the JSON-LD specification [2], clause 6.8 (to the default JSON-LD @context if none available).

6.3.10 Pagination behaviour

For HTTP GET operations (corresponding, in fact, to query-related operations) performed over the resources /entities/, /subscriptions/, /csourceRegistrations/, /csourceSubscriptions/, implementations shall support the HTTP query parameter specified in table 6.3.10-1.

Table 6.3.10-1: Pagination: limit parameter

Name	Data type	Cardinality	Remarks
limit	Positive integer	0..1	It defines the limit to the number of NGSI-LD Elements that shall be retrieved at a maximum as mandated by clause 5.5.9

This clause defines the specific HTTP binding mechanisms that shall be used in conjunction with the behaviours defined by clause 5.5.9. Particularly, to flag the existence of related pages that could be retrieved when dealing with query operations involving pagination, NGSI-LD Systems implementing the HTTP binding shall use the HTTP Link header field as mandated by IETF RFC 8288 [7], clause 3, as follows:

- The pointers to the next and previous pages (when needed as mandated by clause 5.5.9) shall be serialized as link-value elements. The content of such link-value(s) shall be:
  - For the next page, the Link Target shall be a URI-reference that could be dereferenced by an NGSI-LD Client to retrieve the next page of NGSI-LD Elements. In addition, the Link Relation Type shall be equal to "next", registered under the IANA Registry of Link Relation Types [20].
  - For the previous page, the Link Target shall be a URI-reference that could be dereferenced by an NGSI-LD Client to retrieve the previous page of NGSI-LD Elements. In addition, the Link Relation Type shall be equal to "prev", registered under the IANA Registry of Link Relation Types [20].
- At least, the "type" Link Target Attribute shall be included by the previously described serialized Link Header, as mandated by IETF RFC 8288 [7], clause 3.4, and its value shall be exactly equal to the media type resulting from the original request made by the NGSI-LD Client (the request that triggered the current pagination iteration).

EXAMPLE: If the media type requested originally was "application/json" then during the entire pagination iteration the value of the Link Target Attribute "type" shall be "application/json".

6.3.11 Including system-generated attributes

For HTTP GET operations performed over the resources /entities/, /subscriptions/, /csourceRegistrations/, /csourceSubscriptions/ and all of its sub-resources, implementations shall support the parameter specified in table 6.3.11-1.

Table 6.3.11-1: Including system generated attributes: options parameter

Name	Data type	Cardinality	Remarks
options	Comma separated list of strings	0..1	When its value includes the keyword "sysAttrs", a representation of NGSI-LD Elements shall be provided so that the system-generated attributes *createdAt*, *modifiedAt* are included in the response payload body.

6.3.12 Simplified temporal representation of entities

For HTTP GET operations performed over the resource /temporal/entities/ and all of its sub-resources, implementations shall support the parameter specified in table 6.3.12-1.

Table 6.3.12-1: Simplified representation: options parameter

Name	Data type	Cardinality	Remarks
options	Comma separated list of strings	0..1	When its value includes the keyword "temporalValues", a simplified temporal representation of entities shall be provided as defined by clause 4.5.8

6.4 Resource: entities/

6.4.1 Description

This resource represents a collection of entities known to an NGSI-LD system.

6.4.2 Resource definition

Resource URI:

- /entities/

6.4.3 Resource methods

6.4.3.1 POST

This method is bound to the operation "Create Entity" and shall exhibit the behaviour defined by clause 5.6.1, taking the entity to be created from the HTTP request payload body. Figure 6.4.3.1-1 shows the Create Entity interaction and table 6.4.3.1-1 describes the request body and possible responses.

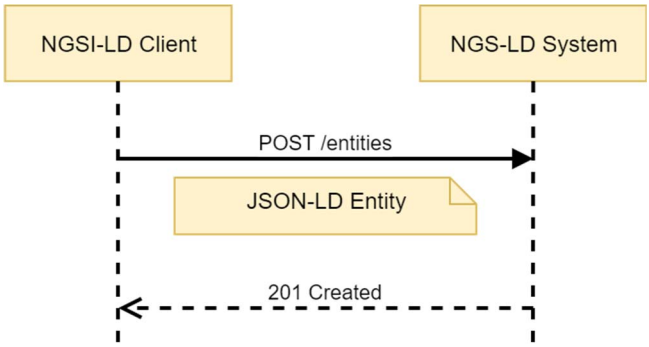

Figure 6.4.3.1-1: Create Entity interaction

Table 6.4.3.1-1: Post Entity request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	NGSI-LD Entity	1	Payload body in the request contains a JSON-LD object which represents the entity that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	Upon success, the HTTP response shall include a "Location" HTTP header that contains the resource URI of the created entity resource.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" member should convey more information about the error.
	ProblemDetails [10]	0..1	409 Already Exists	It is used to indicate that the entity already exists, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	422 Unprocessable Entity	It is used to indicate that the operation is not available, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.4.3.2 GET

This method is associated to the operation "Query Entities" and shall exhibit the behaviour defined by clause 5.7.2, providing entities as part of the HTTP response payload body. Figure 6.4.3.2-1 shows the query entities interaction.

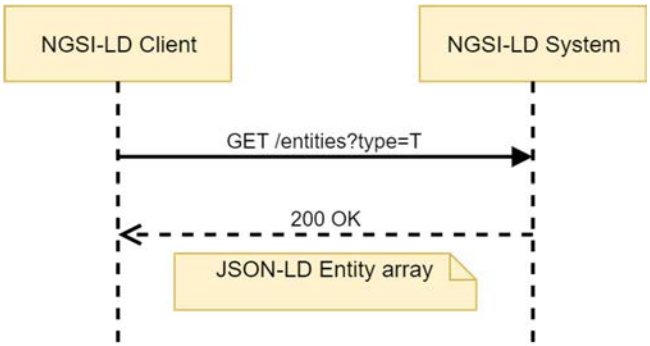

Figure 6.4.3.2-1: Query Entities interaction

The query parameters that shall be supported by implementations are those defined in table 6.4.3.2-1 and table 6.4.3.2-2 describes the request body and possible responses.

Table 6.4.3.2-1: Query parameters

Name	Data type	Cardinality	Remarks
id	Comma separated list of URIs	0..1	List of entity ids to be retrieved
type	Comma separated list of entity types (as short hand string names or URIs)	0..1 It shall be 1 if *attrs* is not present.	List of entity types to be retrieved
idPattern	Regular expression as defined by [11]	0..1	Regular expression that shall be matched by entity ids
attrs	Comma separated list of attribute names (Properties or Relationships)	0..1 It shall be 1 if *type* is not present.	List of Attributes (Properties or Relationships) to be retrieved
q	String	0..1	Query as per clause 4.9
csf	String	0..1	Context Source filter as per clause 4.9
georel	String	0..1 It shall be 1 if *geometry* or *coordinates* are present	Geo relationship as per clause 4.10
geometry	String	0..1 It shall be 1 if *georel* or *coordinates* are present	Geometry as per clause 4.10
coordinates	String	0..1 It shall be one if *georel* or *geometry* are present	Coordinates serialized as a string as per clause 4.10
geoproperty	string representing a Property Name	0..1 It shall be ignored if no geoquery is present	The name of the Property that contains the geospatial data that will be used to resolve the geoquery. By default, will be *location* (see clause 4.7)

Table 6.4.3.2-2: Get Entities request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	NGSI-LD Entity[]	1	200 OK	Upon success, a response body containing the query result as a list of entities.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.5 Resource: entities/{entityId}

6.5.1 Description

This resource represents an entity known to an NGSI-LD system.

6.5.2 Resource definition

Resource URI:

- /entities/{entityId}

Resource URI variables for this resource are defined in table 6.5.2-1.

Table 6.5.2-1: URI variables

Name	Definition
entityId	Id (URI) of the entity to be retrieved

6.5.3 Resource methods

6.5.3.1 GET

This method is associated to the operation "Retrieve Entity" and shall exhibit the behaviour defined by clause 5.7.1. The entity identifier is the value of the resource URI variable "entityId". Figure 6.5.3.1-1 shows the retrieve entity interaction.

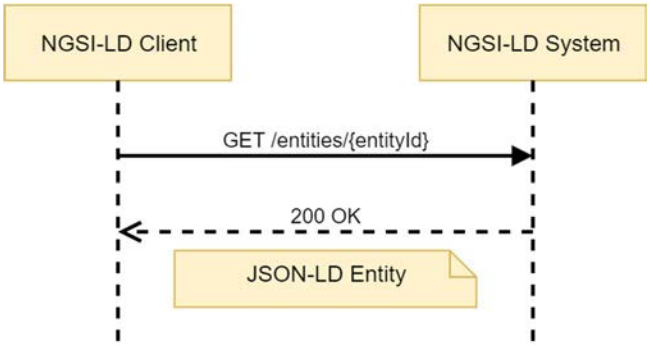

Figure 6.5.3.1-1: Retrieve Entity interaction

The query parameters that shall be supported are those defined in table 6.5.3.1-1 and table 6.5.3.1-2 describes the request body and possible responses.

Table 6.5.3.1-1: Query parameters

Name	Data type	Cardinality	Remarks
attrs	Comma separated list of attribute names (Properties or Relationships)	0..1	List of Attributes to be retrieved. If not specified, all Attributes related to the entity shall be retrieved

Table 6.5.3.1-2: Get Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	NGSI-LD Entity	1	200 OK	Upon success, a response body containing the JSON-LD representation of the target entity containing the selected Attributes.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.5.3.2 DELETE

This method is associated to the operation "Delete Entity" and shall exhibit the behaviour defined by clause 5.6.6. The entity identifier is the value of the resource URI variable "entityId". Figure 6.5.3.2-1 shows the delete entity interaction and table 6.5.3.2-1 describes the request body and possible responses.

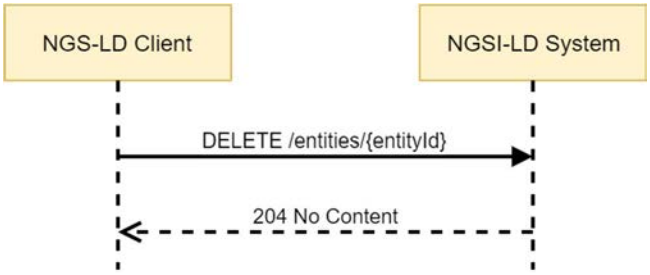

Figure 6.5.3.2-1: Delete Entity interaction

Table 6.5.3.2-1: Delete Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.6 Resource: entities/{entityId}/attrs/

6.6.1 Description

This resource represents all the Attributes (Properties or Relationships) of an NGSI-LD Entity.

6.6.2 Resource definition

Resource URI:

- /entities/{entityId}/attrs

Resource URI variables for this resource are defined in table 6.6.2-1.

Table 6.6.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity

6.6.3 Resource methods

6.6.3.1 POST

This method is bound to the "Append Entity Attributes" operation and shall exhibit the behaviour defined by clause 5.6.3. The entity identifier is the value of the resource URI variable "entityId". The data to be appended shall be contained in the HTTP request payload body. Figure 6.6.3.1-1 shows the append entity attributes interaction and table 6.6.3.1-1 describes the request body and possible responses.

The "options" query parameter for this request can take the following values:

- "noOverwrite". Indicates that no attribute overwrite shall be performed.

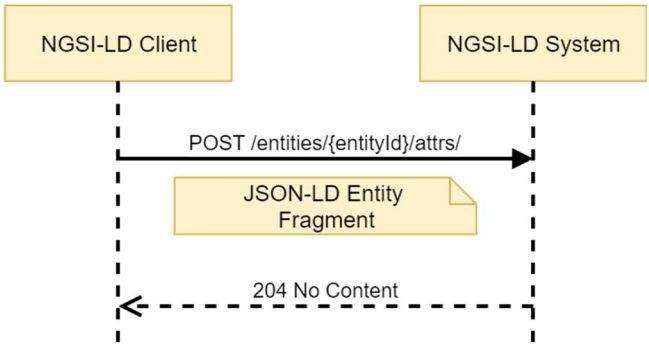

Figure 6.6.3.1-1: Append Entity Attributes interaction

Table 6.6.3.1-1: Post Entity Attributes request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	NGSI-LD Entity Fragment	1	Entity Fragment containing a complete representation of the Attributes to be added.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No content	All the Attributes were appended successfully.
	UpdateResult	1	207 Multi-Status	Only the Attributes included in the response payload body were successfully appended.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.6.3.2 PATCH

This method is bound to the "Update Entity Attributes" operation and shall exhibit the behaviour defined by clause 5.6.2. The entity identifier is the value of the resource URI variable "entityId". The data to be updated shall be contained in the HTTP request payload body. Figure 6.6.3.2-1 shows the Update Entity Attributes interaction and table 6.6.3.2-1 describes the request body and possible responses.

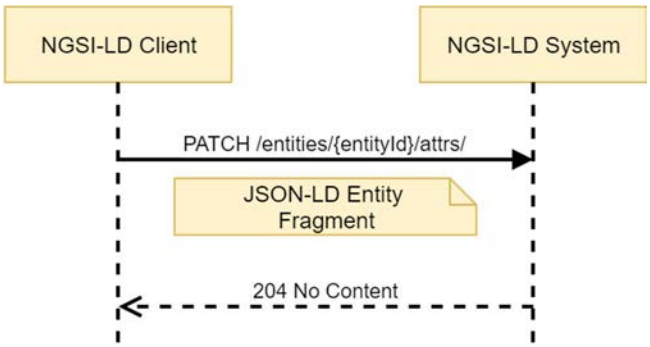

Figure 6.6.3.2-1: Update Entity Attributes interaction

Table 6.6.3.2-1: Patch Entity Attributes request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	NGSI-LD Entity Fragment	1	Entity Fragment containing a complete representation of the Attributes to be updated.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No content	All the Attributes were updated successfully.
	UpdateResult	1	207 Multi-Status	Only the Attributes included in the response payload body were successfully updated. If no Attributes were successfully updated the *updated* array of *UpdateResult* (see clause 5.2.18) will be empty.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier not known to the system, see clause 6.3.2.

6.7 Resource: entities/{entityId}/attrs/{attrId}

6.7.1 Description

This resource represents an attribute (Property or Relationship) of an NGSI-LD Entity.

6.7.2 Resource definition

Resource URI:

- /entities/{entityId}/attrs/{attrId}

Resource URI variables for this resource are defined in table 6.7.2-1.

Table 6.7.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity
attrId	Attribute name (Property or Relationship)

6.7.3 Resource methods

6.7.3.1 PATCH

This method is bound to the "Partial Attribute Update" operation and shall exhibit the behaviour defined by clause 5.6.4. The entity identifier is the value of the resource URI variable "entityId". The attribute name is the value of the resource URI variable "attrId". The Entity Fragment shall be contained in the HTTP request payload body. Figure 6.7.3.1-1 shows the Partial Attribute Update interaction and table 6.7.3.1-1 describes the request body and possible responses.

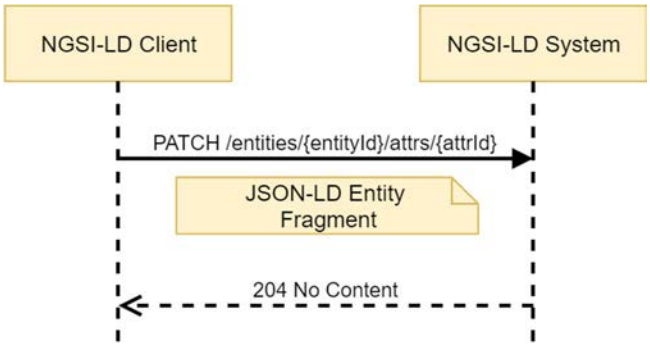

Figure 6.7.3.1-1: Partial Attribute Update interaction

Table 6.7.3.1-1: Patch Entity Attribute request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	NGSI-LD Entity Fragment	1	Entity Fragment containing the elements of the attribute to be updated.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No content	The attribute was updated successfully.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier or attribute name not known to the system, see clause 6.3.2.

6.7.3.2 DELETE

This method is associated to the operation "Delete Entity Attribute" and shall exhibit the behaviour defined by clause 5.6.5. The entity identifier is the value of the resource URI variable "entityId". The attribute name is the value of the resource URI variable "attrId". Figure 6.7.3.2-1 shows the Delete Entity Attribute interaction, table 6.7.3.2-1 shows the delete parameters to be supported and table 6.7.3.2-2 describes the request body and possible responses.

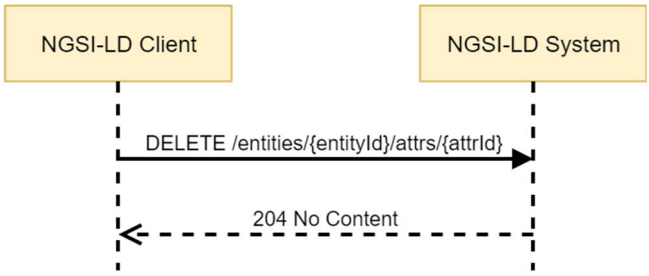

Figure 6.7.3.2-1: Delete Entity Attribute interaction

Table 6.7.3.2-1: Delete parameters

Name	Data type	Cardinality	Remarks
deleteAll	boolean	0..1	If true all attribute instances are deleted, otherwise (default) only attribute instances without a *datasetId* are deleted.
datasetId	URI	0..1	Specifies the *datasetId* of the dataset to be deleted.

Table 6.7.3.2-2: Delete Entity Attribute request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) or attribute name not known to the system. see clause 6.3.2.

6.8 Resource: csourceRegistrations/

6.8.1 Description

This resource represents a collection of context source registrations known to an NGSI-LD system.

6.8.2 Resource definition

Resource URI:

- /csourceRegistrations/

6.8.3 Resource methods

6.8.3.1 POST

This method is bound to the operation "Register Context Source" and shall exhibit the behaviour defined by clause 5.9.2, taking the context source registration to be created from the HTTP request payload body. Figure 6.8.3.1-1 shows the Register Context Source interaction and table 6.8.3.1-1 describes the request body and possible responses.

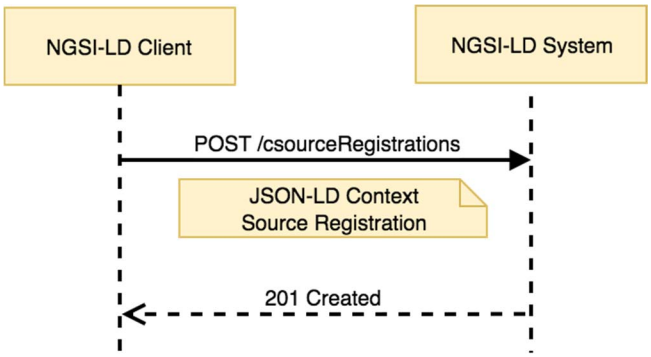

Figure 6.8.3.1-1: Register Context Source interaction

Table 6.8.3.1-1: Patch Attribute request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	CsourceRegistration	1	Payload body in the request contains a JSON-LD object which represents the context source registration that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	Upon success, the HTTP response shall include a "Location" HTTP header that contains the resource URI of the created context source registration resource.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	409 Already Exists	It is used to indicate that the context source registration already exists, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	422 Unprocessable Context Source Registration	It is used to indicate that the operation is not available see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.8.3.2 GET

This method is associated to the operation "Query Context Source Registrations" and shall exhibit the behaviour defined by clause 5.10.2, i.e. the parameters in the request describe entity related information, but instead of directly providing this entity information, the context source registration data, which describes context sources that can possibly provide the information, are returned as part of the HTTP response payload body. Figure 6.8.3.2-1 shows the Query Context Source Registrations interaction.

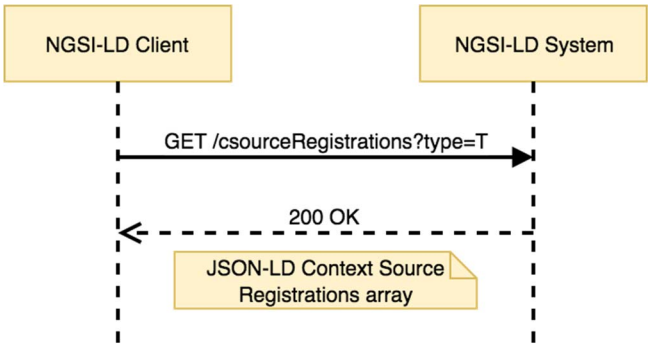

Figure 6.8.3.2-1: Query Context Source Registrations interaction

The query parameters that shall be supported by implementations are those defined in table 6.8.3.2-1 and table 6.8.3.2-2 describes the request body and possible responses.

Table 6.8.3.2-1: Query parameters

Name	Data type	Cardinality	Remarks
id	Comma separated list of URIs	0..1	List of entity ids to be retrieved
type	Comma separated list of entity types as short hand string names or URIs	0..1	List of entity types to be retrieved
idPattern	Regular expression as defined by [11]	0..1	Regular expression that shall be matched by entity ids satisfying the query
attrs	Comma separated list of attribute names (Properties or Relationships)	0..1	List of Attributes (Properties or Relationships) to be retrieved
q	String	0..1	Query as per clause 4.9
csf	String	0..1	Context Source filter as per clause 4.9
georel	String	0..1 It shall be 1 if "geometry" or "coordinates" are present	Geo relationship as per clause 4.10
geometry	String	0..1 It shall be 1 if "georel" or "coordinates" are present	Geometry as per clause 4.10
coordinates	String	0..1 It shall be one if "georel" or "geometry" are present	Coordinates serialized as a string as per clause 4.10
geoproperty	string representing a Property name	0..1 It shall be ignored if no geoquery is present	The name of the Property that contains the geospatial data that will be used to resolve the geoquery
timeproperty	string representing a Property name	0..1 It shall be ignored if no temporal query is present	The name of the Property that contains the temporal data that will be used to resolve the temporal query
timerel	String representing the temporal relationship as defined by clause 4.11	0..1	Allowed values: "before", "after", "between"
time	String representing the *time* parameter as defined by clause 4.11	0..1	It shall be a *DateTime*. Cardinality shall be 1 if *timerel* is present.
endTime	String representing the *endTime* parameter as defined by clause 4.11	0..1	It shall be a *DateTime.* Cardinality shall be 1 if *timerel* is equal to "between"

Table 6.8.3.2-2: Get Context Source Registrations request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	CSourceRegistration[]	1	200 OK	Upon success, a response body containing the query result as an array of context source registrations.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.9 Resource: csourceRegistrations/{registrationId}

6.9.1 Description

This resource represents a collection of context source registrations known to an NGSI-LD system.

6.9.2 Resource definition

Resource URI:

- /csourceRegistrations/{registrationId}

Resource URI variables for this resource are defined in table 6.9.2-1.

Table 6.9.2-1: URI variables

Name	Definition
registrationId	Id (URI) of the context source registration

6.9.3 Resource methods

6.9.3.1 GET

This method is associated with the operation "Retrieve Context Source Registration" and shall exhibit the behaviour defined by clause 5.10.1. The registration identifier is the value of the resource URI variable "registrationId". Figure 6.9.3.1-1 shows the Retrieve Context Source Registration interaction and table 6.9.3.1-1 describes the request body and possible responses.

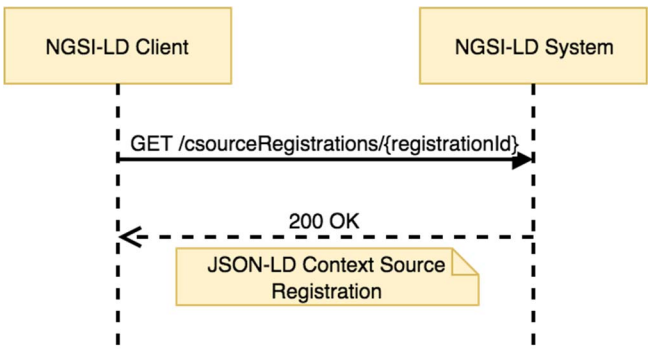

Figure 6.9.3.1-1: Retrieve Context Source Registration interaction

Table 6.9.3.1-1: Get Context Source Registration request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	CsourceRegistration	1	200 OK	Upon success, a response body containing the JSON-LD representation of the target context source registration.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an context source registration identifier (URI) not known to the system, see clause 6.3.2.

6.9.3.2 PATCH

This method is bound to the "Update Context Source Registration" operation and shall exhibit the behaviour defined by clause 5.9.3. The context source registration identifier is the value of the resource URI variable "registrationId". The context source registration to be updated shall be contained in the HTTP request payload body. Figure 6.9.3.2-1 shows the Update Context Source Registration interaction and table 6.9.3.2-1 describes the request body and possible responses.

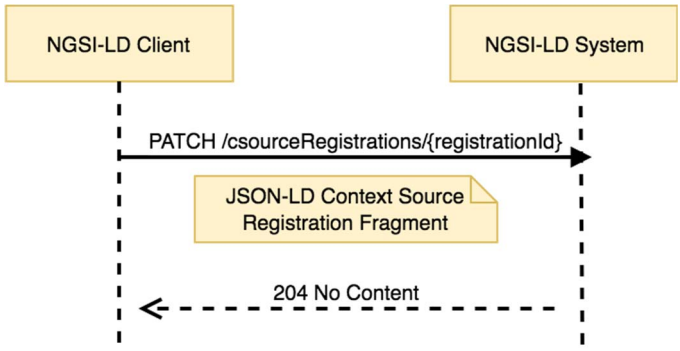

Figure 6.9.3.2-1: Update Context Source Registration interaction

Table 6.9.3.2-1: Patch Context Source Registration request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	CSourceRegistration	1	Payload body in the request contains a JSON-LD object which represents the context source registration that is to be updated.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No content	The context source registration was successfully updated.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a context source registration identifier not known to the system, see clause 6.3.2.

6.9.3.3 DELETE

This method is associated to the operation "Delete Context Source Registration" and shall exhibit the behaviour defined by clause 5.9.4. The context source registration identifier is the value of the resource URI variable "registrationId". Figure 6.9.3.3-1 shows the Delete Context Source Registration interaction and table 6.9.3.3-1 describes the request body and possible responses.

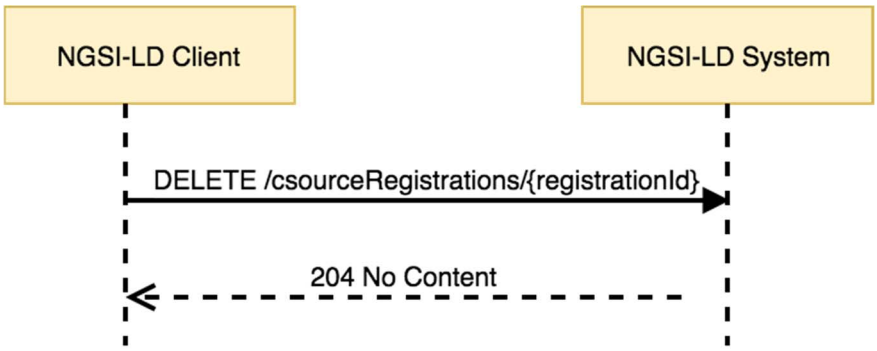

Figure 6.9.3.3-1: Delete Context Source Registration interaction

Table 6.9.3.3-1: Delete Context Source Registration request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a context source registration identifier (URI) not known to the system, see clause 6.3.2.

6.10 Resource: subscriptions/

6.10.1 Description

This resource represents a collection of subscriptions known to an NGSI-LD system.

6.10.2 Resource definition

Resource URI:

- /subscriptions/

6.10.3 Resource methods

6.10.3.1 POST

This method is bound to the operation "Create Subscription" and shall exhibit the behaviour defined by clause 5.8.1, taking the subscription to be created from the HTTP request payload body. Figure 6.10.3.1-1 shows the Create Subscription interaction and table 6.10.3.1-1 describes the request body and possible responses.

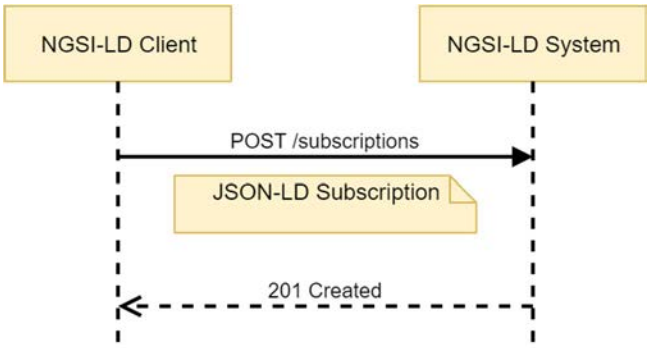

Figure 6.10.3.1-1: Create Subscription interaction

Table 6.10.3.1-1: Post Subscription request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Subscription	1	Payload body in the request contains a JSON-LD object which represents the subscription that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	Upon success, the HTTP response shall include a "Location" HTTP header that contains the resource URI of the created subscription resource.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	409 Already Exists	It is used to indicate that the subscription already exists see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.10.3.2 GET

This method is associated to the operation "Query Subscriptions" and shall exhibit the behaviour defined by clause 5.8.4, providing the subscription data as part of the HTTP response payload body. Figure 6.10.3.2-1 shows the Query Subscriptions interaction.

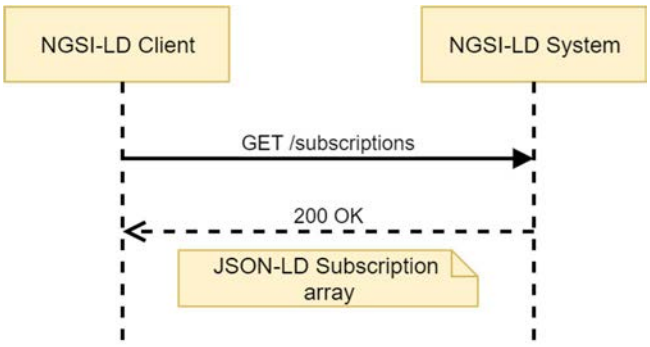

Figure 6.10.3.2-1: Query Subscriptions interaction

The query parameters that shall be supported by implementations are those defined in table 6.10.3.2-1 and table 6.10.3.2-2 describes the request body and possible responses.

Table 6.10.3.2-1: Query parameters

Name	Data type	Cardinality	Remarks
limit	Number	0..1	Maximum number of subscriptions to be retrieved

Table 6.10.3.2-2: Get Subscriptions request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription[]	1	200 OK	Upon success, a response body containing a list of subscriptions.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.11 Resource: subscriptions/{subscriptionId}

6.11.1 Description

This resource represents a subscription known to an NGSI-LD system.

6.11.2 Resource definition

Resource URI:

- /subscriptions/{subscriptionId}

Resource URI variables for this resource are defined in table 6.11.2-1.

Table 6.11.2-1: URI variables

Name	Definition
subscriptionId	Id (URI) of the concerned subscription

6.11.3 Resource methods

6.11.3.1 GET

This method is associated to the operation "Retrieve Subscription" and shall exhibit the behaviour defined by clause 5.8.3. The subscription identifier is the value of the resource URI variable "subscriptionId". Figure 6.11.3.1-1 shows the Retrieve Subscription interaction and table 6.11.3.1-1 describes the request body and possible responses.

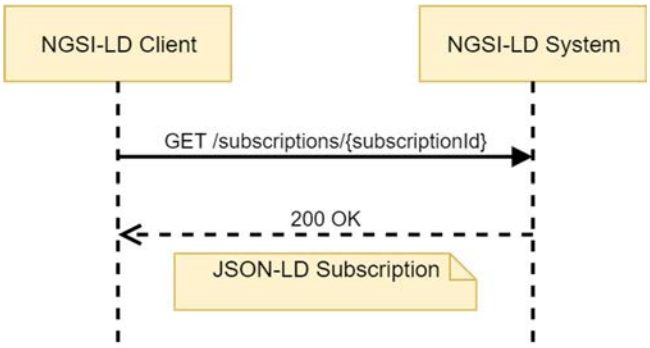

Figure 6.11.3.1-1: Retrieve Subscription interaction

Table 6.11.3.1-1: Get Subscription request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription	1	200 OK	Upon success, a response body containing the JSON-LD representation of the target subscription.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.11.3.2 PATCH

This method is associated to the operation "Update Subscription" and shall exhibit the behaviour defined by clause 5.8.2. The subscription identifier is the value of the resource URI variable "subscriptionId". Figure 6.11.3.2-1 shows the Update Subscription interaction and table 6.11.3.2-1 describes the request body and possible responses.

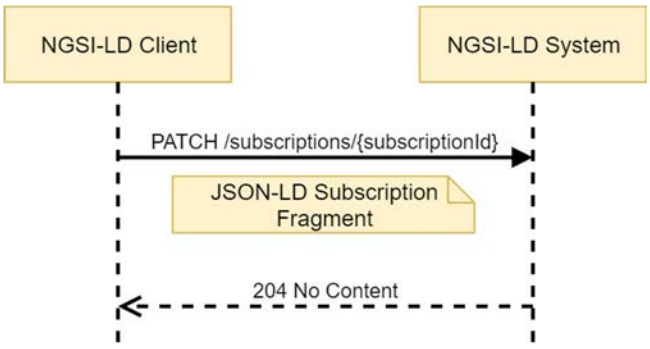

Figure 6.11.3.2-1: Update Subscription interaction

Table 6.11.3.2-1: Patch Subscription request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	Subscription Fragment	1	Subscription Fragment including id, type and any other subscription field to be changed	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.11.3.3 DELETE

This method is associated to the operation "Delete Subscription" and shall exhibit the behaviour defined by clause 5.8.5. The subscription identifier is the value of the resource URI variable "subscriptionId". Figure 6.11.3.3-1 shows the Delete Subscription interaction and table 6.11.3.3-1 describes the request body and possible responses.

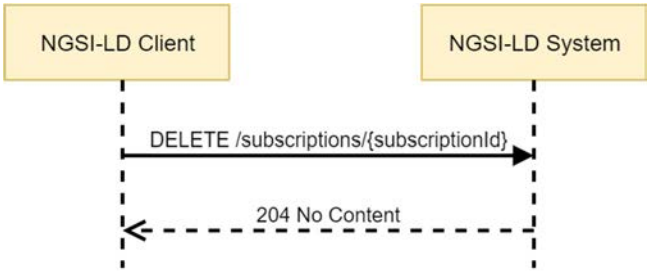

Figure 6.11.3.3-1: Delete Subscription interaction

Table 6.11.3.3-1: Delete Subscription request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.12 Resource: csourceSubscriptions/

6.12.1 Description

This resource represents a collection of context source registration subscriptions known to an NGSI-LD system.

6.12.2 Resource definition

Resource URI:

- /csourceSubscriptions/

6.12.3 Resource methods

6.12.3.1 POST

This method is bound to the operation "Create Context Source Registration Subscription" and shall exhibit the behaviour defined by clause 5.11.2, taking the context source registration subscription to be created from the HTTP request payload body. Figure 6.12.3.1-1 shows the Create Context Source Registration Subscription interaction and table 6.12.3.1-1 describes the request body and possible responses.

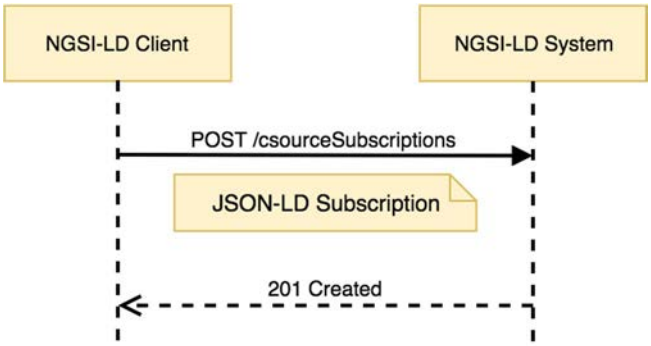

Figure 6.12.3.1-1: Create Context Source Registration Subscription interaction

Table 6.12.3.1-1: Post Context Source Registration Subscription request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	Subscription	1	Payload body in the request contains a JSON-LD object which represents the context source registration subscription that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	Upon success, the HTTP response shall include a "Location" HTTP header that contains the resource URI of the created context source registration subscription resource.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	409 Already Exists	It is used to indicate that the context source registration subscription already exists, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.12.3.2 GET

This method is associated to the operation "Query Context Source Registration Subscriptions" and shall exhibit the behaviour defined by clause 5.11.5, providing the context source registration subscription data as part of the HTTP response payload body. Figure 6.12.3.2-1 shows the Query Context Source Registration Subscriptions interaction.

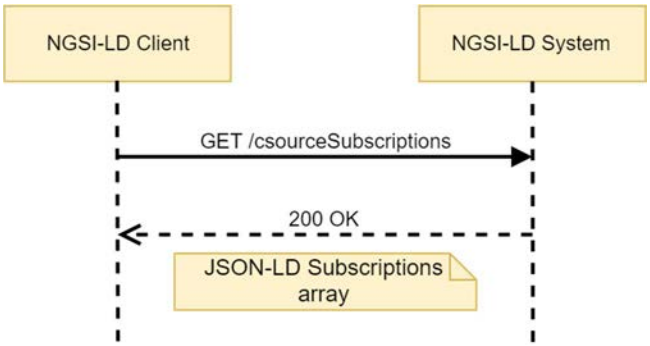

Figure 6.12.3.2-1: Query Context Source Registration Subscriptions interaction

The query parameters that shall be supported by implementations are those defined in table 6.12.3.2-1 and table 6.12.3.2-2 describes the request body and possible responses.

Table 6.12.3.2-1: Query parameters

Name	Data type	Cardinality	Remarks
limit	Number	0..1	Maximum number of subscriptions to be retrieved

Table 6.12.3.2-2: Get Context Source Registration Subscriptions request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription[]	1	200 OK	Upon success, a response body containing a list of context source registration subscriptions.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.13 Resource: csourceSubscriptions/{subscriptionId}

6.13.1 Description

This resource represents a context source registration subscription known to an NGSI-LD system.

6.13.2 Resource definition

Resource URI:

- /csourceSubscriptions/{subscriptionId}

Resource URI variables for this resource are defined in table 6.13.2-1.

Table 6.13.2-1: URI variables

Name	Definition
subscriptionId	Id (URI) of the concerned context source registration subscription

6.13.3 Resource methods

6.13.3.1 GET

This method is associated to the operation "Retrieve Context Source Registration Subscription" and shall exhibit the behaviour defined by clause 5.11.4. The subscription identifier is the value of the resource URI variable "subscriptionId". Figure 6.13.3.1-1 shows the Retrieve Context Source Registration interaction and table 6.13.3.1-1 describes the request body and possible responses.

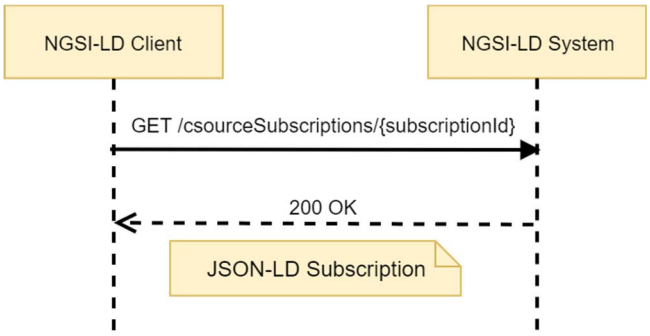

Figure 6.13.3.1-1: Retrieve Context Source Registration Subscription interaction

Table 6.13.3.1-1: Get Context Source Registration Subscription request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription	1	200 OK	Upon success, a response body containing the JSON-LD representation of the target context source registration subscription.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.13.3.2 PATCH

This method is associated to the operation "Update Context Source Registration Subscription" and shall exhibit the behaviour defined by clause 5.11.3. The subscription identifier is the value of the resource URI variable "subscriptionId". Figure 6.13.3.2-1 shows the Update Context Source Registration Subscription interaction and table 6.13.3.2-1 describes the request body and possible responses.

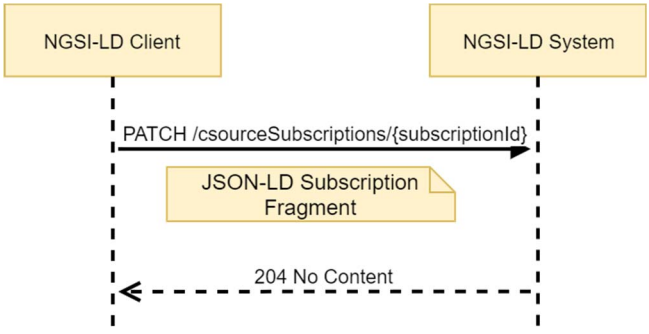

Figure 6.13.3.2-1: Update Context Source Registration Subscription interaction

Table 6.13.3.2-1: Patch Context Source Registration Subscription request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	Subscription Fragment	1	Subscription Fragment including id, type and any other context source registration subscription field to be changed	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.13.3.3 DELETE

This method is associated to the operation "Delete Context Source Registration Subscription" and shall exhibit the behaviour defined by clause 5.11.6. The subscription identifier is the value of the resource URI variable "subscriptionId". Figure 6.13.3.3-1 shows the Delete Context Source Registration Subscription interaction and table 6.13.3.3-1 describes the request body and possible responses.

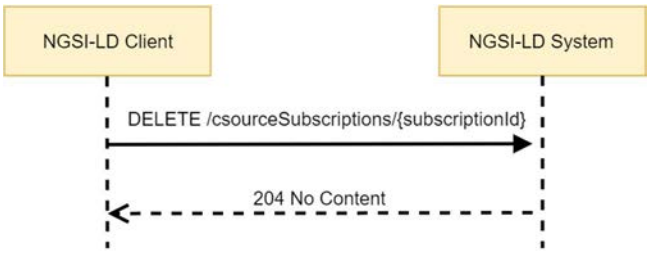

Figure 6.13.3.3-1: Delete Context Source Registration Subscription interaction

Table 6.13.3.3-1: Delete Context Source Registration Subscription request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.14 Resource: entityOperations/create

6.14.1 Description

A sub-resource, pertaining to the *entityOperations/* resource, intended to enable batch entity creation for the NGSI-LD API.

6.14.2 Resource definition

Resource URI:

- /entityOperations/create

6.14.3 Resource methods

6.14.3.1 POST

This method is associated to the operation "Batch Entity Creation" and shall exhibit the behaviour defined by clause 5.6.7. Figure 6.14.3.1-1 shows the operation interaction and table 6.14.3.1-1 describes the request body and possible responses.

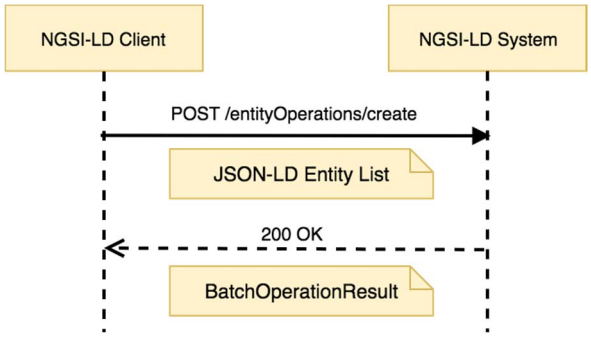

Figure 6.14.3.1-1: Batch Entity Creation Interaction

Table 6.14.3.1-1: Batch Entity Creation Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of entities to be created	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	BatchOperationResult	1	200 OK	Upon success, a response body containing the result of each operation contained in the batch.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.15 Resource: entityOperations/upsert

6.15.1 Description

A sub-resource, pertaining to the *entityOperations/* resource, intended to enable batch entity creation or update for the NGSI-LD API.

6.15.2 Resource definition

Resource URI:

- /entityOperations/upsert

6.15.3 Resource methods

6.15.3.1 POST

This method is associated to the operation "Batch Entity Creation or Update (Upsert)" and shall exhibit the behaviour defined by clause 5.6.8. Figure 6.15.3.1-1 shows the operation interaction and table 6.15.3.1-1 describes the request body and possible responses.

The "options" query parameter for this request can take the following values:

- "replace". Indicates that all the existing Entity content shall be replaced (default mode).
- "update". Indicates that existing Entity content shall be updated.

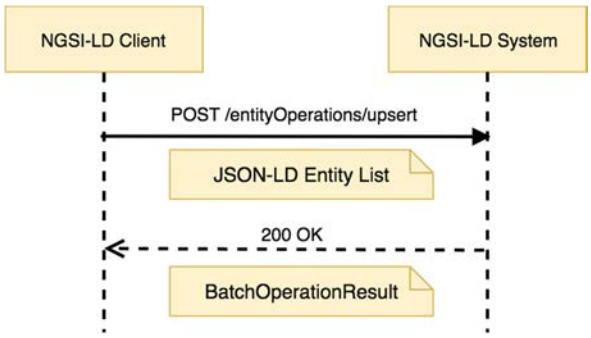

Figure 6.15.3.1-1: Batch Entity Creation or Update Interaction

Table 6.15.3.1-1: Batch Entity Creation or Update Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of entities to be created / updated	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	BatchOperationResult	1	200 OK	Upon success, a response body containing the result of each operation contained in the batch.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.16 Resource: entityOperations/update

6.16.1 Description

A sub-resource, pertaining to the *entityOperations*/ resource, intended to enable batch entity update for the NGSI-LD API.

6.16.2 Resource definition

Resource URI:

- /entityOperations/update

6.16.3 Resource methods

6.16.3.1 POST

This method is associated to the operation "Batch Entity Update" and shall exhibit the behaviour defined by clause 5.6.9. Figure 6.16.3.1-1 shows the operation interaction and table 6.16.3.1-1 describes the request body and possible responses.

The "options" query parameter for this request can take the following values:

- "noOverwrite". Indicates that no attribute overwrite shall be performed.

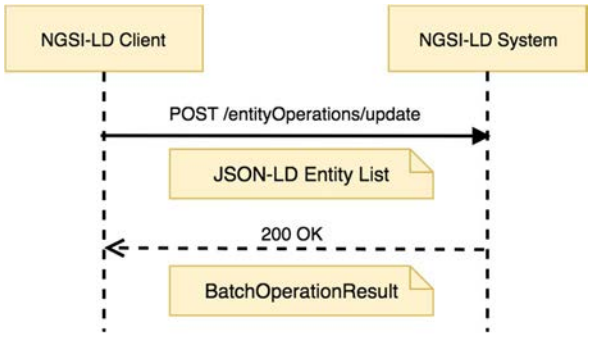

Figure 6.16.3.1-1: Batch Entity Update Interaction

Table 6.16.3.1-1: Batch Entity Update Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of Entities to be updated	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	BatchOperationResult	1	200 OK	Upon success, a response body containing the result of each operation contained in the batch.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.17 Resource: entityOperations/delete

6.17.1 Description

A sub-resource, pertaining to the *entityOperations*/ resource, intended to enable batch entity deletion for the NGSI-LD API.

6.17.2 Resource definition

Resource URI:

- /entityOperations/delete

6.17.3 Resource methods

6.17.3.1 POST

This method is associated to the operation "Batch Entity Delete" and shall exhibit the behaviour defined by clause 5.6.10. Figure 6.17.3.1-1 shows the operation interaction and table 6.17.3.1-1 describes the request body and possible responses.

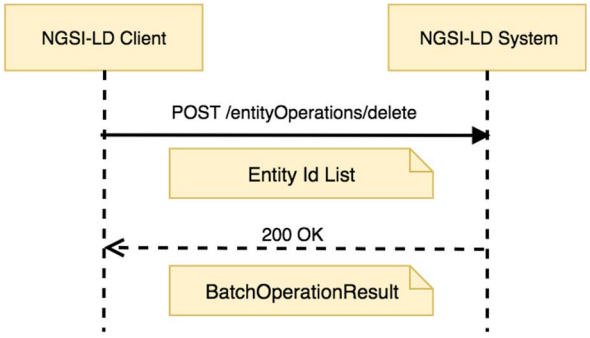

Figure 6.17.3.1-1: Batch Entity Delete Interaction

Table 6.17.3.1-1: Batch Entity Update Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	URI[]	1	Array of Entity Ids corresponding to the Entities to be deleted	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	BatchOperationResult	1	200 OK	Upon success, a response body containing the result of each operation contained in the batch.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.18 Resource: temporal/entities/

6.18.1 Description

This resource represents the temporal evolution of Entities known to an NGSI-LD system.

6.18.2 Resource definition

Resource URI:

- /temporal/entities/

6.18.3 Resource methods

6.18.3.1 POST

This method is associated to the operation "Create or Update Temporal Representation of Entities" and shall exhibit the behaviour defined by clause 5.6.11, taking the temporal representation of entity to be created from the HTTP request payload body. Figure 6.18.3.1-1 shows this interaction (for creation) and table 6.18.3.1-1 describes the request body and possible responses.

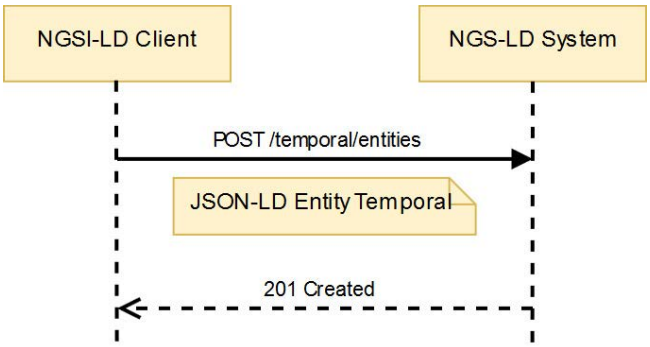

Figure 6.18.3.1-1: Create Temporal Representation of Entity interaction

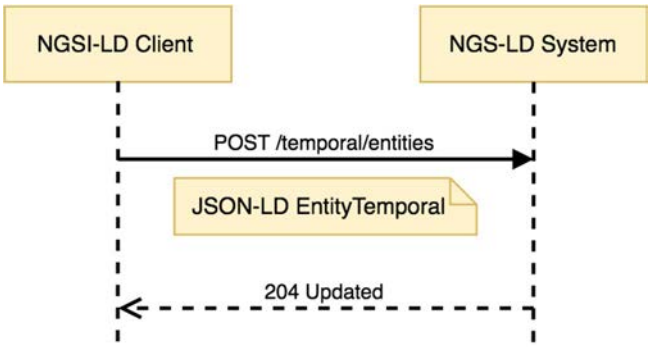

Figure 6.18.3.1-2: Update Temporal Representation of Entity interaction

Table 6.18.3.1-1: Post EntityTemporal request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	NGSI-LD EntityTemporal	1	Payload body in the request contains a JSON-LD object which represents the temporal representation of the entity that is to be created (or updated).	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	Upon creation success, the HTTP response shall include a "Location" HTTP header that contains the resource URI of the created entity resource.
			204 No Content	Upon update success
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" member should convey more information about the error.
	ProblemDetails [10]	0..1	422 Unprocessable Entity	It is used to indicate that the operation is not available, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.18.3.2 GET

This method is associated to the operation "Query Temporal Evolution of Entities" and shall exhibit the behaviour defined by clause 5.7.4, providing the temporal evolution of the matching Entities as part of the HTTP response payload body. Figure 6.18.3.2-1 shows this interaction.

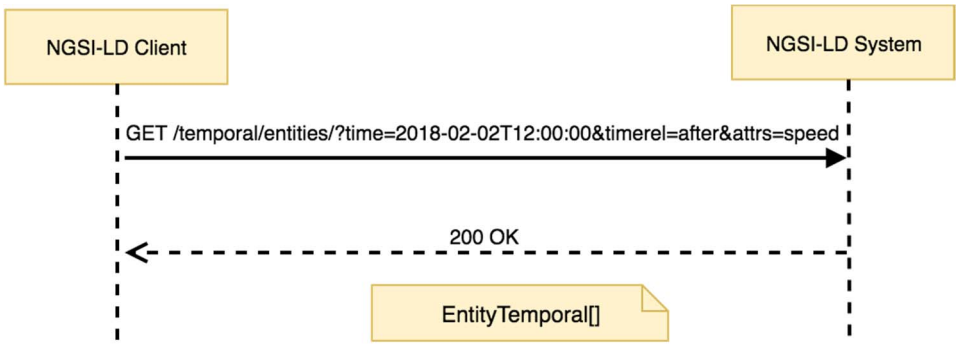

Figure 6.18.3.2-1: Query Temporal Evolution of Entities interaction

The query parameters that shall be supported by implementations are those defined in table 6.18.3.2-1 and table 6.18.3.2-2 describes the request body and possible responses.

Table 6.18.3.2-1: Temporal Evolution Query parameters

Name	Data type	Cardinality	Remarks
id	Comma separated list of URIs	0..1	List of entity ids to be retrieved
type	Comma separated list of entity type names	0..1 It shall be 1 if *attrs* is not present	List of entity types to be retrieved
idPattern	Regular expression as defined by [11]	0..1	Regular expression that shall be matched by entity ids
attrs	Comma separated list of attribute names (Properties or Relationships)	0..1 It shall be 1 if *type* is not present	List of Attributes (Properties or Relationships) to be retrieved
q	String	0..1	Query as per clause 4.9
csf	String	0..1	Context Source filter as per clause 4.9
georel	String	0..1 It shall be 1 if *geometry* or *coordinates* are present	Geo relationship as per clause 4.10
geometry	String	0..1 It shall be 1 if *georel* or *coordinates* are present	Geometry as per clause 4.10
coordinates	String	0..1 It shall be one if *georel* or *geometry* are present	Coordinates serialized as a string as per clause 4.10
geoproperty	String representing a Property Name	0..1 It shall be ignored if no geo-query is present	The name of the Property that contains the geospatial data that will be used to resolve the geo-query. By default, will be *location*. (See clause 4.7)
timeproperty	String representing a Property Name	0..1	The name of the Property that contains the temporal data that will be used to resolve the temporal query. By default, will be *observedAt*. (See clause 4.8)
timerel	String representing the temporal relationship as defined by clause 4.11	1	Allowed values: "before", "after", "between"
time	String representing the *time* parameter as defined by clause 4.11	1	It shall be a *DateTime*
endTime	String representing the *endTime* parameter as defined by clause 4.11	0..1	It shall be a *DateTime.* Cardinality shall be 1 if *timerel* is equal to "between"
lastN	Positive integer	0..1	Only the last n instances, per Attribute, per Entity (under the specified time interval) shall be retrieved

Table 6.18.3.2-2: Query Entities History request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	NGSI-LD EntityTemporal[]	1	200 OK	Upon success, a response body containing the query result as a list of temporal representation of Entities.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.19 Resource: temporal/entities/{entityId}

6.19.1 Description

This resource is associated to the temporal representation of an Entity known to an NGSI-LD system.

6.19.2 Resource definition

Resource URI:

- /temporal/entities/{entityId}

Resource URI variables for this resource are defined in table 6.19.2-1.

Table 6.19.2-1: URI variables

Name	Definition
entityId	Id (URI) of the entity to be retrieved

6.19.3 Resource methods

6.19.3.1 GET

This method is associated to the operation "Retrieve temporal evolution of an Entity" and shall exhibit the behaviour defined by clause 5.7.3. The Entity identifier is the value of the resource URI variable *entityId*. Figure 6.19.3.1-1 shows the retrieve temporal representation of an entity interaction.

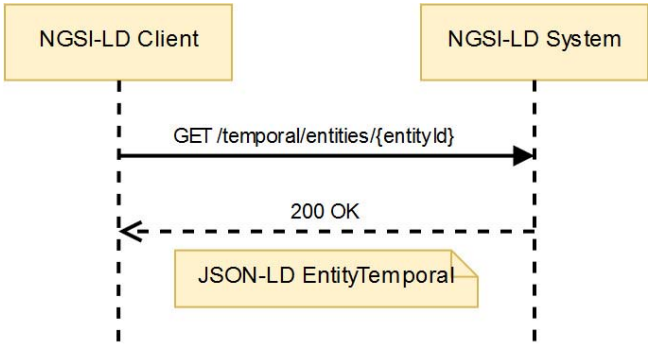

Figure 6.19.3.1-1: Retrieve Temporal evolution of an Entity interaction

The query parameters that shall be supported are those defined in table 6.19.3.1-1 and table 6.19.3.1-2 describes the request body and possible responses.

Table 6.19.3.1-1: Query parameters

Name	Data type	Cardinality	Remarks
attrs	Comma separated list of attribute names (Properties or Relationships)	0..1	List of Attributes to be retrieved. If not specified, all Attributes related to the temporal representation of an entity shall be retrieved.
timeproperty	String representing a Property Name	0..1	The name of the Property that contains the temporal data that will be used to resolve the temporal query. By default, will be *observedAt* (see clause 4.8).
timerel	String representing the temporal relationship as defined by clause 4.11	0 It shall be 1 if *time* is present	Allowed values: "before", "after", "between".
time	String representing the *time* parameter as defined by clause 4.11	0..1 It shall be 1 if *timerel* is present	It shall be a *DateTime*.
endTime	String representing the *endTime* parameter as defined by clause 4.11	0..1 It shall be 1 if *timerel* is equal to "between"	It shall be a *DateTime.*
lastN	Positive integer	0..1	Only the last n Attribute instances (under the concerned time interval) shall be retrieved.

Table 6.19.3.1-2: Get Temporal Representation of Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	NGSI-LD EntityTemporal	1	200 OK	Upon success, a response body containing the JSON-LD temporal representation of the target entity containing the selected Attributes.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.19.3.2 DELETE

This method is associated to the operation "Delete Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.16. The Entity identifier is the value of the resource URI variable *entityId*.
Figure 6.19.3.2-1 shows the delete entity interaction and table 6.19.3.2-1 describes the request body and possible responses.

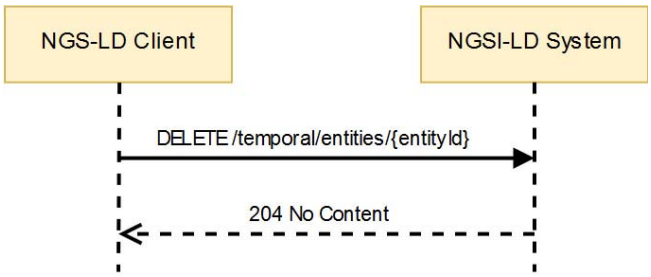

Figure 6.19.3.2-1: Delete Temporal Representation of Entity interaction

Table 6.19.3.2-1: Delete Temporal Representation of Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.20 Resource: temporal/entities/{entityId}/attrs/

6.20.1 Description

This resource represents all the Attributes (Properties or Relationships) of a Temporal Representation of an NGSI-LD Entity.

6.20.2 Resource definition

Resource URI:

- /temporal/entities/{entityId}/attrs/

Resource URI variables for this resource are defined in table 6.20.2-1.

Table 6.20.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity

6.20.3 Resource methods

6.20.3.1 POST

This method is bound to the "Add Attributes to Temporal Representation of an Entity" operation and shall exhibit the behaviour defined by clause 5.6.12. The Entity identifier is the value of the resource URI variable *entityId*. The data to be added shall be contained in the HTTP request payload body. Figure 6.20.3.1-1 shows the add entity attributes interaction and table 6.20.3.1-1 describes the request body and possible responses.

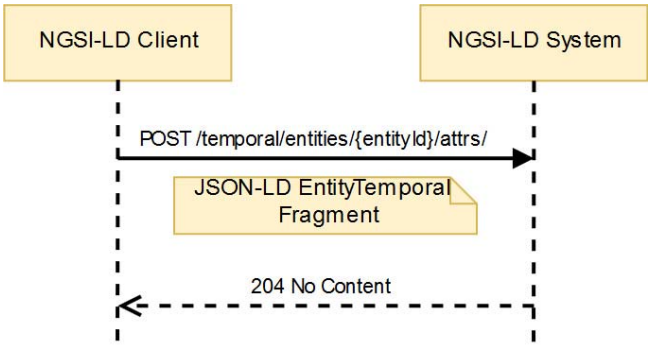

Figure 6.20.3.1-1: Add Attributes to Temporal Representation of an Entity interaction

Table 6.20.3.1-1: Post Add Attributes to Temporal Representation of an Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	NGSI-LD EntityTemporal Fragment	1	EntityTemporal Fragment containing a complete representation of the Attribute instances to be added.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No content	All the Attributes were added successfully.
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.21 Resource: temporal/entities/{entityId}/attrs/{attrId}

6.21.1 Description

This resource represents an Attribute (Property or Relationship) of a Temporal Representation of an NGSI-LD Entity.

6.21.2 Resource definition

Resource URI:

- /temporal/entities/{entityId}/attrs/{attrId}

Resource URI variables for this resource are defined in table 6.21.2-1.

Table 6.21.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity
attrId	Attribute name (Property or Relationship)

6.21.3 Resource methods

6.21.3.1 DELETE

This method is associated to the operation "Delete Attribute from Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.13. The Entity identifier is the value of the resource URI variable *entityId*. The Attribute Name is the value of the resource URI variable *attrId*. Figure 6.21.3.1-1 shows the Delete Attribute from Temporal Representation of an Entity interaction, table 6.21.3.1-1 shows the delete parameters to be supported and table 6.21.3.1-2 describes the request body and possible responses.

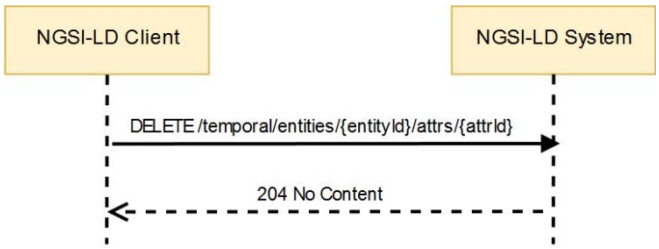

Figure 6.21.3.1-1: Delete Attribute from Temporal Representation of an Entity interaction

Table 6.21.3.1-1: Delete parameters

Name	Data type	Cardinality	Remarks
deleteAll	boolean	0..1	If true all attribute instances are deleted, otherwise (default) only attribute instances without a *datasetId* are deleted.
datasetId	URI	0..1	Specifies the *datasetId* of the dataset to be deleted.

Table 6.21.3.1-2: Delete Attribute from Temporal Representation of an Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI) or Attribute Name not known to the system. See clause 6.3.2.

6.22 Resource: temporal/entities/{entityId}/attrs/{attrId}/{instanceId}

6.22.1 Description

This resource represents an Attribute (Property or Relationship) instance of a Temporal Representation of an NGSI-LD Entity.

6.22.2 Resource definition

Resource URI:

- /temporal/entities/{entityId}/attrs/{attrId}/{instanceId}

Resource URI variables for this resource are defined in table 6.22.2-1.

Table 6.22.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity
attrId	Attribute Name (Property or Relationship)
instanceId	Id (URI) identifying a particular Attribute instance

6.22.3 Resource methods

6.22.3.1 PATCH

This method is associated to the operation "Modify attribute instance from Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.14. The Entity identifier is the value of the resource URI variable *entityId*. The attribute name is the value of the resource URI variable *attrId*. The instance identifier is the value of the resource URI variable *instanceId*. Figure 6.22.3.1-1 shows the Modify Entity Attribute instance interaction and table 6.22.3.1-1 describes the request body and possible responses.

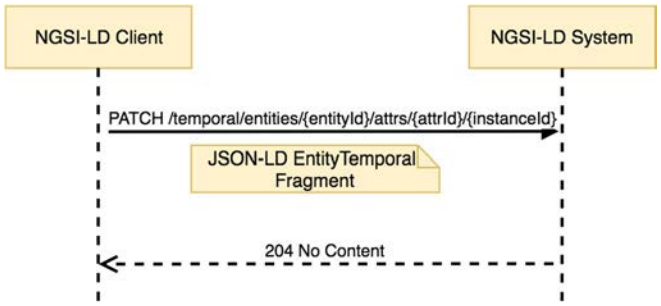

Figure 6.22.3.1-1: Modify Entity Attribute instance from Temporal Representation interaction

Table 6.22.3.1-1: Modify Entity Attribute instance from Temporal Representation request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI), attribute name or instance identifier not known to the system. See clause 6.3.2.

6.22.3.2 DELETE

This method is associated to the operation "Delete Attribute instance from Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.15. The Entity identifier is the value of the resource URI variable *entityId*. The Attribute Name is the value of the resource URI variable *attrId*. The instance identifier is the value of the resource URI variable *instanceId*. Figure 6.22.3.2-1 shows the Delete Entity Attribute instance interaction and table 6.22.3.2-1 describes the request body and possible responses.

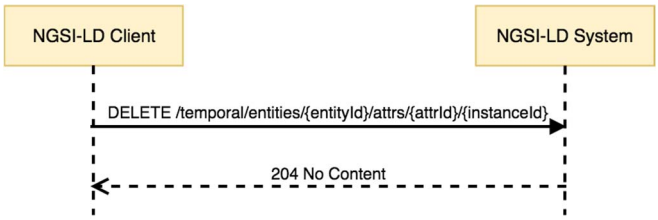

Figure 6.22.3.2-1: Delete Entity Attribute instance from Temporal Representation interaction

Table 6.22.3.2-1: Delete Entity Attribute instance from Temporal Representation request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A			
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A		204 No Content	
	ProblemDetails [10]	0..1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.
	ProblemDetails [10]	0..1	404 Not Found	It is used when a client provided an entity identifier (URI), attribute name or instance identifier not known to the system. See clause 6.3.2.

Annex A (normative): NGSI-LD identifier considerations

A.1 Introduction

The purpose of identifiers is to allow uniquely identifying NGSI-LD elements (Entities, Context Subscriptions or Context Source Registrations) within an NGSI-LD system. This annex is intended to clarify the different issues around the design of identifiers in NGSI-LD.

A.2 Entity identifiers

In order to enable the participation of NGSI-LD in linked data scenarios, all Entities are identified by **URIs**. If those URIs are expected to participate in external linked data relationships they **should** be dereferenceable.

It is noteworthy that the identifier from the point of view of NGSI-LD is different from the inherent identifier that a specific Entity may have. For instance, an NGSI-LD Entity of Type *Vehicle* may have a Property named *licencePlateNumber*, which it is actually a unique identifier from the point of view of the Entity domain, as it uniquely identifies the specific vehicle instance. However, from the point of view of the NGSI-LD system, it may have another identifier which might or might not include such licence plate number identifier.

A.3 NGSI-LD namespace

NGSI-LD defines a specific URN [9] namespace intended to help API users to design readable, clean and simple identifiers. As it is based on URNs, the usage of this identification approach is not recommended when dereferenceable URIs are needed (fully-fledged linked data scenarios).

The referred namespace is defined as follows (to be registered with IANA):

- Namespace identifier: NID = "ngsi-ld"
- Namespace specific string: NSS = EntityTypeName ":" EntityIdentificationString

EntityTypeName shall be an Entity Type Name which can be expanded to a URI as per the @context.

EntityIdentificationString shall be a string that allows uniquely identifying the subject Entity in combination with the other items being part of the NSS.

EXAMPLE: urn:ngsi-ld:**Person**:28976543.

It is recommended that applications use this URN namespace when applicable.

Annex B (normative): Core NGSI-LD @context definition

Below it is the definition of the Core NGSI-LD @context which shall be supported by implementations.

Such definition has been tested using [i.7].

```
{
    "@context": {
      "ngsi-ld": "https://uri.etsi.org/ngsi-ld/",
      "id": "@id",
      "type": "@type",
      "value": "https://uri.etsi.org/ngsi-ld/hasValue",
      "object": {
        "@id": "https://uri.etsi.org/ngsi-ld/hasObject",
        "@type":"@id"
      },
      "Property": "https://uri.etsi.org/ngsi-ld/Property",
      "Relationship": "https://uri.etsi.org/ngsi-ld/Relationship",
      "DateTime": "https://uri.etsi.org/ngsi-ld/DateTime",
      "Date": "https://uri.etsi.org/ngsi-ld/Date",
      "Time": "https://uri.etsi.org/ngsi-ld/Time",
      "createdAt": {
        "@id": "https://uri.etsi.org/ngsi-ld/createdAt",
        "@type": "DateTime"
      },
      "modifiedAt": {
        "@id": "https://uri.etsi.org/ngsi-ld/modifiedAt",
        "@type": "DateTime"
      },
      "observedAt": {
        "@id": "https://uri.etsi.org/ngsi-ld/observedAt",
        "@type": "DateTime"
      },
      "datasetId": {
        "@id": "https://uri.etsi.org/ngsi-ld/datasetId",
        "@type": "@id"
      },
      "instanceId": {
        "@id": "https://uri.etsi.org/ngsi-ld/instanceId",
        "@type": "@id"
      },
      "unitCode": "https://uri.etsi.org/ngsi-ld/unitCode",
      "location": "https://uri.etsi.org/ngsi-ld/location",
      "observationSpace": "https://uri.etsi.org/ngsi-ld/observationSpace",
      "operationSpace": "https://uri.etsi.org/ngsi-ld/operationSpace",
      "GeoProperty": "https://uri.etsi.org/ngsi-ld/GeoProperty",
      "TemporalProperty": "https://uri.etsi.org/ngsi-ld/TemporalProperty",
      "ContextSourceRegistration": "https://uri.etsi.org/ngsi-ld/ContextSourceRegistration",
      "Subscription": "https://uri.etsi.org/ngsi-ld/Subscription",
      "Notification": "https://uri.etsi.org/ngsi-ld/Notification",
      "ContextSourceNotification": "https://uri.etsi.org/ngsi-ld/ContextSourceNotification",
      "title": "https://uri.etsi.org/ngsi-ld/title",
      "detail": "https://uri.etsi.org/ngsi-ld/detail",
      "idPattern": "https://uri.etsi.org/ngsi-ld/idPattern",
      "name": "https://uri.etsi.org/ngsi-ld/name",
      "description": "https://uri.etsi.org/ngsi-ld/description",
      "information": "https://uri.etsi.org/ngsi-ld/information",
      "observationInterval": "https://uri.etsi.org/ngsi-ld/observationInterval",
      "managementInterval": "https://uri.etsi.org/ngsi-ld/managementInterval",
      "expires": {
        "@id": "https://uri.etsi.org/ngsi-ld/expires",
        "@type": "DateTime"
      },
      "endpoint": "https://uri.etsi.org/ngsi-ld/endpoint",
      "entities": "https://uri.etsi.org/ngsi-ld/entities",
      "properties": {
        "@id": "https://uri.etsi.org/ngsi-ld/properties",
        "@type": "@vocab"
      },
      "relationships": {
        "@id": "https://uri.etsi.org/ngsi-ld/relationships",
        "@type": "@vocab"
      },
      "start": {
```

```
  "@id": "https://uri.etsi.org/ngsi-ld/start",
  "@type": "DateTime"
},
"end": {
  "@id": "https://uri.etsi.org/ngsi-ld/end",
  "@type": "DateTime"
},
"watchedAttributes":{
  "@id": "https://uri.etsi.org/ngsi-ld/watchedAttributes",
  "@type": "@vocab"
},
"timeInterval": "https://uri.etsi.org/ngsi-ld/timeInterval",
"q": "https://uri.etsi.org/ngsi-ld/q",
"geoQ": "https://uri.etsi.org/ngsi-ld/geoQ",
"csf": "https://uri.etsi.org/ngsi-ld/csf",
"isActive": "https://uri.etsi.org/ngsi-ld/isActive",
"notification": "https://uri.etsi.org/ngsi-ld/notification",
"status": "https://uri.etsi.org/ngsi-ld/status",
"throttling": "https://uri.etsi.org/ngsi-ld/throttling",
"temporalQ": "https://uri.etsi.org/ngsi-ld/temporalQ",
"geometry": "https://uri.etsi.org/ngsi-ld/geometry",
"coordinates": "https://uri.etsi.org/ngsi-ld/coordinates",
"georel": "https://uri.etsi.org/ngsi-ld/georel",
"geoproperty": "https://uri.etsi.org/ngsi-ld/geoproperty",
"attributes": {
  "@id": "https://uri.etsi.org/ngsi-ld/attributes",
  "@type": "@vocab"
},
"format": "https://uri.etsi.org/ngsi-ld/format",
"timesSent": "https://uri.etsi.org/ngsi-ld/timesSent",
"lastNotification":{
  "@id": "https://uri.etsi.org/ngsi-ld/lastNotification",
  "@type": "DateTime"
},
"lastFailure":{
  "@id": "https://uri.etsi.org/ngsi-ld/lastFailure ",
  "@type": "DateTime"
},
"lastSuccess":{
  "@id": "https://uri.etsi.org/ngsi-ld/lastSuccess",
  "@type": "DateTime"
},
"uri": "https://uri.etsi.org/ngsi-ld/uri",
"accept": "https://uri.etsi.org/ngsi-ld/accept",
"success": {
  "@id": "https://uri.etsi.org/ngsi-ld/success",
  "@type": "@id"
},
"errors": "https://uri.etsi.org/ngsi-ld/errors",
"error": "https://uri.etsi.org/ngsi-ld/error",
"entityId": {
  "@id": "https://uri.etsi.org/ngsi-ld/entityId",
  "@type": "@id"
},
"updated": "https://uri.etsi.org/ngsi-ld/updated",
"unchanged": "https://uri.etsi.org/ngsi-ld/unchanged",
"attributeName": "https://uri.etsi.org/ngsi-ld/attributeName",
"reason": "https://uri.etsi.org/ngsi-ld/reason",
"timerel": "https://uri.etsi.org/ngsi-ld/timerel",
"time": {
  "@id": "https://uri.etsi.org/ngsi-ld/time",
  "@type": "DateTime"
},
"endTime": {
  "@id": "https://uri.etsi.org/ngsi-ld/endTime",
  "@type": "DateTime"
},
"timeproperty": "https://uri.etsi.org/ngsi-ld/timeproperty",
"subscriptionId": {
  "@id": "https://uri.etsi.org/ngsi-ld/subscriptionId",
  "@type": "@id"
},
"notifiedAt":{
  "@id": "https://uri.etsi.org/ngsi-ld/notifiedAt",
  "@type": "DateTime"
},
"data": "https://uri.etsi.org/ngsi-ld/data",
"triggerReason": "https://uri.etsi.org/ngsi-ld/triggerReason",
```

```
      "values":{
          "@id": "https://uri.etsi.org/ngsi-ld/hasValues",
          "@container": "@list"
      },
      "objects":{
          "@id": "https://uri.etsi.org/ngsi-ld/hasObjects",
          "@type": "@id",
          "@container": "@list"
      },
      "@vocab": "https://uri.etsi.org/ngsi-ld/default-context/"
    }
}
```

NOTE 1: Implementers can take advantage of prefixed terms, i.e. in the form ngsi-ld:term, to provide a terser representation of the Core @context.

NOTE 2: This API specification has defined the Core @context terms "name", "description" and "title" to have the same meaning as the https://schema.org/name, https://schema.org/description and https://schema.org/title terms (https://schema.org).

Annex C (informative): Examples of using the API

C.1 Introduction

This annex is informative and is intended to show in action the JSON-LD representation defined by NGSI-LD.

JSON representations of the examples shown in this annex can be found at [i.15].

C.2 Entity Representation

C.2.1 Property Graph

Figure C.2.1-1 shows a diagram representing a property graph to be used for the examples discussed in this clause.

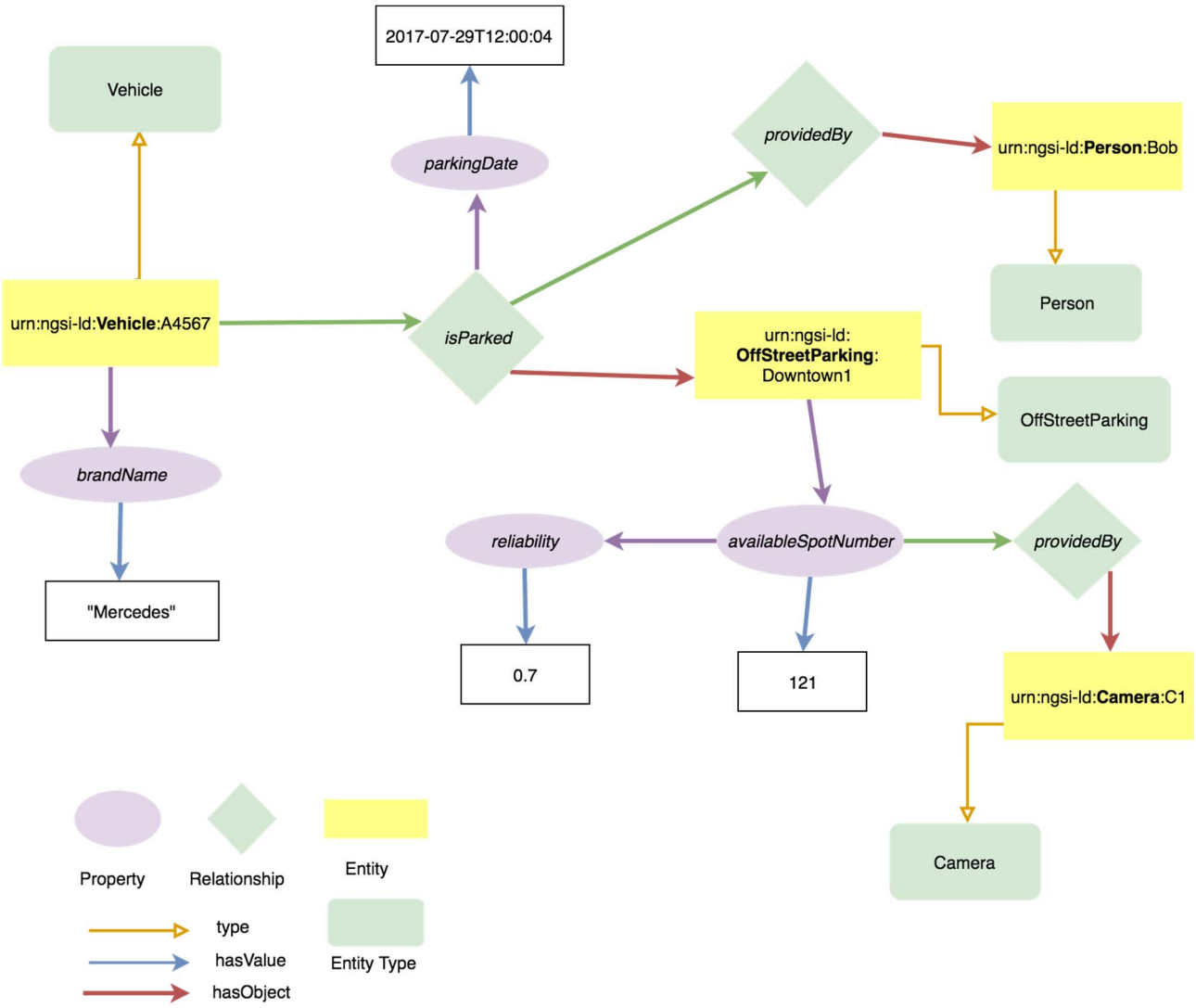

Figure C.2.1-1: Reference example

As per the algorithms described above and as per the rules for generating the JSON-LD representation of NGSI-LD entities the above graph will result in the following JSON-LD representations. The syntax has been checked using the JSON-LD Playground tool [i.5].

C.2.2 Vehicle Entity

Below there is a representation of an Entity of Type "Vehicle". It can be observed that the @context is composed of different parts, namely the Core @context and several vocabulary-specific @contexts.

It is noteworthy that the @context corresponding to the Parking domain is included as it is referenced through the *isParked* Relationship.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": {
        "type": "Property",
        "value": "Mercedes"
    },
    "isParked": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:OffStreetParking:Downtown1",
        "observedAt": "2017-07-29T12:00:04Z",
        "providedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Person:Bob"
        }
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

Simplified representation

The simplified representation is a collapsed representation of an Entity, which focuses on Property Values and Relationship objects present at the first level of the graph.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": "Mercedes",
    "isParked": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

Multiple attribute example

Below is an example, where the speed of the car is provided by two different sources. As both may be relevant at the same time, there are two individual attribute instances for speed; each is identified by a *datasetId* and is represented in NGSI-LD by an Attribute Name with an index. The *datasetId* enables individually creating, updating and deleting a particular instance without affecting the instance from another source.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "speed#1": {
        "type": "Property",
        "value": 55,
        "source": {
            "type": "Property",
            "value": "Speedometer"
        },
        "datasetId": "urn:ngsi-ld:Property:speedometerA4567-speed"
```

```
    },
    "speed#2": {
        "type": "Property",
        "value": 54.5,
        "source": {
            "type": "Property",
            "value": "GPS"
        },
        "datasetId": "urn:ngsi-ld:Property:gpsBxyz123-speed"
    },
      "@context": [
        {
            "speed#1": "http://example.org/speed",
            "speed#2": "http://example.org/speed",
            "source": "http://example.org/hasSource"
        },
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

C.2.3 Parking Entity

Below there is a representation of an Entity of Type "OffStreetParking". It can be observed that the @context is composed of two different elements, the Core one and the vocabulary-specific one.

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "OffStreetParking",
    "name": {
        "type": "Property",
        "value": "Downtown One"
    },
    "availableSpotNumber": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z",
        "reliability": {
            "type": "Property",
            "value": 0.7
        },
        "providedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Camera:C1"
        }
    },
    "totalSpotNumber": {
        "type": "Property",
        "value": 200
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [-8.5, 41.2]
        }
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

Simplified representation

The Simplified Representation (a.k.a. keyValues) is a collapsed representation of an Entity, which focuses on Property Values and Relationship objects present at the first level of the graph.

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "OffStreetParking",
    "name": "Downtown One",
    "availableSpotNumber": 121,
    "totalSpotNumber": 200,
    "location": {
        "type": "Point",
```

```
        "coordinates": [-8.5, 41.2]
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

C.2.4 @context

The disposition of the @context can be as an inline JSON object, as a dereferenceable URI or as a (multiple) combination of both. In the examples above the @context is provided through several dereferenceable URIs. The resulting @context (obtained by merging the content of the resource referenced by the referred URIs) is shown below.

NOTE 1: For brevity reasons the @context does not contain the API terms defined by clause 5.2.

NOTE 2: Some extra terms are defined because they will be used in examples later presented.

```
{
    "id": "@id",
    "type": "@type",
    "Property": "https://uri.etsi.org/ngsi-ld/Property",
    "Relationship": "https://uri.etsi.org/ngsi-ld/Relationship",
    "value": "https://uri.etsi.org/ngsi-ld/hasValue",
    "object": {
        "@type": "@id",
        "@id": "https://uri.etsi.org/ngsi-ld/hasObject"
    },
    "observedAt": {
        "@type": "https://uri.etsi.org/ngsi-ld/DateTime",
        "@id": "https://uri.etsi.org/ngsi-ld/observedAt"
    },
    "datasetId": {
        "@id": "https://uri.etsi.org/ngsi-ld/datasetId",
        "@type":"@id"
    },
    "location": "https://uri.etsi.org/ngsi-ld/location",
    "GeoProperty": "https://uri.etsi.org/ngsi-ld/GeoProperty",
    "Vehicle": "http://example.org/vehicle/Vehicle",
    "brandName": "http://example.org/vehicle/brandName",
    "speed": "http://example.org/vehicle/speed",
    "isParked": {
        "@type": "@id",
        "@id": "http://example.org/common/isParked"
    },
    "OffStreetParking": "http://example.org/parking/OffStreetParking",
    "availableSpotNumber": "http://example.org/parking/availableSpotNumber",
    "totalSpotNumber": "http://example.org/parking/totalSpotNumber",
    "isNextToBuilding": {
        "@type": "@id",
        "@id": "http://example.org/common/isNextToBuilding"
    },
    "reliability": "http://example.org/common/reliability",
    "providedBy": {
        "@type": "@id",
        "@id": "http://example.org/common/providedBy"
    },
    "name": "http://example.org/common/name"
}
```

C.3 Context Source Registration

Below there is an example representation of a Context Source Registration. It makes use of the @context formerly described.

```
{
    "id": "urn:ngsi-ld:ContextSourceRegistration:csr1a3456",
    "type": "ContextSourceRegistration",
    "information": [
        {
        "entities": [
```

```
            {
                "id": "urn:ngsi-ld:Vehicle:A456",
                "type": "Vehicle"
            }
        ],
        "properties": ["brandName","speed"],
        "relationships": ["isParked"]
        },
        {
        "entities": [
            {
                "idPattern": ".*downtown$",
                "type": "OffStreetParking"
            },
            {
                "idPattern": ".*47$",
                "type": "OffStreetParking"
             }
        ],
        "properties": ["availableSotNumber","totalSpotNumber"],
        "relationships": ["isNextToBuilding"]
        }
    ],
    "endpoint": "http://my.csource.org:1026",
    "location": {
        "type": "Polygon",
        "coordinates": [
             [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
               [100.0, 1.0], [100.0, 0.0]] ]
     },
     "timestamp": {
        "start": " 2017-11-29T14:53:15Z"
     },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

The Registration is referring to a Context Source capable of providing information from Entities of type *Vehicle* and *OffStreetParking*, meeting certain id requirements. More concretely, it can only provide the referenced Properties and Relationships. In addition, the Registration example covers a particular geographical area and a temporal scope which starts at a point in time.

C.4 Context Subscription

Below there is an example of a Context Subscription. It makes use of the @context formerly described.

```
{
    "id": "urn:ngsi-ld:Subscription:mySubscription",
    "type": "Subscription",
    "entities": [
        {
            "type": "Vehicle"
        }
    ],
    "watchedAttributes": ["speed"],
    "q": "speed>50",
    "geoQ": {
        "georel": "near;maxDistance==2000",
        "geometry": "Point",
         "coordinates": [-1,100]
    },
    "notification": {
        "attributes": ["speed"],
        "format": "keyValues",
        "endpoint": {
            "uri": "http://my.endpoint.org/notify",
            "accept": "application/json"
        }
    },
    "@context": [
```

```
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
       "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

The subject of the Context Subscription are Entities of Type *Vehicle* which *speed* is greater than 50, and located close to a certain area defined by a reference spatial point. Every time the *speed* (watched Attribute) of a concerned vehicle, changes, a new notification (including the new speed value) will be received in the specified endpoint.

C.5 HTTP REST API Examples

C.5.1 Introduction

This clause introduces some simple usage examples of the NGSI-LD API (HTTP REST binding). They are not intended to be exhaustive but just a sample for helping readers to understand better the present document. Nonetheless, it is the intention of ETSI ISG CIM to publish in the near future a Developer's Primer with much more examples.

C.5.2 Create Entity of Type Vehicle

C.5.2.1 HTTP Request

POST /ngsi-ld/v1/entities/

Content-Type: application/ld+json

Content-Length: 556

<Insert Here the JSON-LD representation of a Vehicle as described by clause C.2.2 Vehicle Entity>

C.5.2.2 HTTP Response

201 Created

Location: /ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:A4567

C.5.3 Query Entities

C.5.3.1 Introduction

Please give me all the Entities of type *Vehicle* which brand name is not "Mercedes". Only tell me the brand name and please provide the data in the NGSI-LD Simplified Format.

C.5.3.2 HTTP Request

GET /ngsi-ld/v1/entities/?type=Vehicle&q=brandName!=Mercedes&options=keyValues

Accept: application/ld+json

Link: <http://example.org/ngsi-ld/latest/aggregatedContext.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

C.5.3.3 HTTP Response

200 OK

Content-Type: application/ld+json


```
[
    {
        "id": "urn:ngsi-ld:Vehicle:B9211",
        "type": "Vehicle",
        "brandName": "Volvo",
        "@context": [
            "http://example.org/ngsi-ld/latest/vehicle.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
        ]
    }
]
```


C.5.4 Query Entities (Pagination)

C.5.4.1 Introduction

Please give me all the Entities of type *Vehicle*. Only tell me the brand name attribute and please provide the data in the NGSI-LD Simplified Format. Limit the number of entities retrieved to 2.

C.5.4.2 HTTP Request

GET /ngsi-ld/v1/entities/?type=Vehicle&options=keyValues&limit=2

Accept: application/ld+json

Link: <http://example.org/ngsi-ld/latest/aggregatedContext.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

C.5.4.3 HTTP Response

200 OK

Content-Type: application/ld+json

Link: </ngsi-ld/v1/entities/?type=Vehicle&options=keyValues&limit=2&offset=2>; rel="next"; type="application/ld+json"


```
[
    {
        "id": "urn:ngsi-ld:Vehicle:B9211",
        "type": "Vehicle",
        "brandName": "Volvo",
        "@context": [
            "http://example.org/ngsi-ld/latest/vehicle.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
        ]
    },
    {
        "id": "urn:ngsi-ld:Vehicle:A456",
        "type": "Vehicle",
        "brandName": "Mercedes",
        "@context": [
            "http://example.org/ngsi-ld/latest/vehicle.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
        ]
    }
]
```

C.5.5 Temporal Query

C.5.5.1 Introduction

Please give me all the temporal evolution of the attribute "speed" of Entities of type *Vehicle* which brand name is not "Mercedes" between the 1st of August at noon and the 1st of August at 01 PM.

C.5.5.2 HTTP Request

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&q=brandName!=Mercedes&attrs=speed,brandName&timerel=between&time=2018-08-01:12:00:00Z&endTime=2018-08-01:13:00:00Z

Accept: application/ld+json

Link: <http://example.org/ ngsi-ld /latest/aggregatedContext.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

C.5.5.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
    {
        "id": "urn:ngsi-ld:Vehicle:B9211",
        "type": "Vehicle",
        "brandName": [
            {
                "type": "Property",
                "value": "Volvo",
            }
        ],
        "speed": [
            {
                "type": "Property",
                "value": 120,
                "observedAt": "2018-08-01T12:03:00Z"
            },
            {
                "type": "Property",
                "value": 80,
                "observedAt": "2018-08-01T12:05:00Z"
            },
            {
                "type": "Property",
                "value": 100,
                "observedAt": "2018-08-01T12:07:00Z"
            }
        ],
        "@context": [
            "http://example.org/ngsi-ld/latest/vehicle.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
        ]
    }
]
```

C.5.6 Temporal Query (Simplified Representation)

C.5.6.1 Introduction

Please give me all the speed history of Entities of type *Vehicle* which brand name is not "Mercedes" between the 1st of August at noon and the 1st of August at 01 PM. Simplified representation is required.

C.5.6.2 HTTP Request

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&q=brandName!=Mercedes&attrs=speed,brandName&timerel=between&time=2018-08-01:12:00:00Z&endTime=2018-08-01:13:00:00Z&options=**temporalValues**

Accept: application/ld+json

Link: <http://example.org/ngsi-ld/latest/aggregatedContext.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

C.5.6.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
    {
        "id": "urn:ngsi-ld:Vehicle:B9211",
        "type": "Vehicle",
        "brandName": {
            "type": "Property",
            "values": [
                        ["Volvo",""]
                      ]
        },
        "speed": {
                "type": "Property",
                "values": [
                            [120,"2018-08-01T12:03:00Z"],
                            [80,"2018-08-01T12:05:00Z"],
                            [100,"2018-08-01T12:07:00Z"]
                          ]
        },
        "@context": [
            "http://example.org/ngsi-ld/latest/vehicle.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
        ]
    }
]
```

C.6 Date Representation

The following example shows how to represent time values (*Date*, *Time*, or *DateTime*) in NGSI-LD using the syntax offered by JSON-LD. User-defined Properties whose value is a time value (*Date*, *DateTime or Time*) are defined as *Property*, not as *TemporalProperty*, and are serialized in NGSI-LD use the *@value* syntax structure, as shown by the example below.

```
{
      "id": "urn:ngsi-ld:Vehicle:B9211",
      "type": "Vehicle",
      "testedAt": {
        "type": "Property",
        "value": {
          "@type": "DateTime",
          "@value": "2018-12-04T12:00:00Z"
        }
      },
      "@context": [
          "http://example.org/ngsi-ld/latest/vehicle.jsonld",
          "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
      ]
}
```

In addition, it is recommended that in the @context JSON-LD declaration of Properties which value is a time value to include a declaration of the form:

```
"testedAt": {
        "@type": "https://uri.etsi.org/ngsi-ld/DateTime",
        "@id": "http://example.org/test/P1"
}
```

For simplicity reasons, a *TemporalProperty* is represented only by its Value, i.e. no Properties of *TemporalProperty* nor Relationships of *TemporalProperty* can be conveyed. In more formal language, a *TemporalProperty* does not allow reification. It is important to remark that the term *TemporalProperty* has been reserved for the semantic tagging of non-reified structural timestamps (*observedAt*, *createdAt*, *modifiedAt*), which capture the temporal evolution of Entity Attributes. Only such structural timestamps can be used as *timeproperty* in Temporal Queries as mandated by clause 4.11.

C.7 @context utilization clarifications

When expanding or compacting JSON-LD terms, the JSON-LD @context to be used is always the one provided in the current API request. For the benefit of users and implementers the following examples illustrate this concept:

The scenario starts with the creation of an Entity using a JSON-LD @context as follows:

POST /ngsi-ld/v1/entities/

Content-Type: application/ld+json

Content-Length: 200

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "OffStreetParking",
    "name": {
        "type": "Property",
        "value": "Downtown One"
    },
    "availableSpotNumber": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    },
    "totalSpotNumber": {
        "type": "Property",
        "value": 200
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

The content of the @context utilized for the referred Entity creation (at http://example.org/ngsi-ld/latest/parking.jsonld) is as follows:

```
{
   "OffStreetParking": "http://example.org/parking/OffStreetParking",
   "availableSpotNumber": "http://example.org/parking/availableSpotNumber",
   "totalSpotNumber": "http://example.org/parking/totalSpotNumber",
}
```

At Entity creation time the implementation will perform the expansion of terms using the JSON-LD @context depicted above.

Now it is needed to retrieve our initial Entity. For retrieving such Entity, this time, a different JSON-LD @context is going to be utilized, as follows:

```
{
   "OffP": "http://example.org/parking/OffStreetParking",
   "ava": "http://example.org/parking/availableSpotNumber",
   "total": "http://example.org/parking/totalSpotNumber",
}
```

This new @context, even though it makes use of the same set of Fully Qualified Names, is defining new short strings as terms. The reasons for that could be to multiple: to facilitate data consumption by clients, to save some bandwidth, to enable a more (or less) human-readable response payload body in a language different than English, etc.

In this particular case, the result of the Entity retrieval will be as depicted below. It can be observed that the terms defined by the JSON-LD @context **provided at retrieval time** are used to render the Entity content (compaction), and **not** the terms that were provided at creation time (which may be no longer known by the NGSI-LD Broker).

It is also interesting to note that the @context array of the response payload body contains, indeed, our header-supplied @context.

GET /ngsi-ld/v1/entities/urn:ngsi-ld:OffStreetParking:Downtown1

Accept: application/ld+json

Link: <http://example.org/ngsi-ld/latest/parking-abbreviated.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "OffP",
    "name": {
        "type": "Property",
        "value": "Downtown One"
    },
    "ava": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    },
    "total": {
        "type": "Property",
        "value": 200,
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/parking-abbreviated.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

Another interesting case to note is the one when an @context with no matching terms or no @context at all is supplied. See the following example:

GET /ngsi-ld/v1/entities/urn:ngsi-ld:OffStreetParking:Downtown1

Accept: application/ld+json

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "http://example.org/parking/OffStreetParking",
    "name": {
        "type": "Property",
        "value": "Downtown One"
    },
    "http://example.org/parking/availableSpotNumber": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    },
    "http://example.org/parking/totalSpotNumber": {
        "type": "Property",
        "value": 200,
    },
    "@context": "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
}
```

In this particular case it can be observed that the user names (Entity Type, Attributes) in the response payload body have not been compacted, and as a result the Fully Qualified Names are included. However, the core API terms have been compacted, as the Core @context is always implicit (and that is why it is included in the JSON-LD response payload body, as mandated by the specification). Please note that the term "name" has been compacted as it is part of the Core @context.

C.8 Link header utilization clarifications

The JSON-LD Specification [2] states clearly that **only one HTTP Link header** with the link relationship <http://www.w3.org/ns/json-ld#context> is required to appear. Such statement has implications in terms of providing the JSON-LD @context when using the NGSI-LD API. The main implication is that if the @context is a compound one, i.e. an @context which references multiple individual @context, served by resources behind different URIs, then a **wrapper** @context has to be created and hosted. The final aim is that only one @context is referenced from the JSON-LD Link header. This can be illustrated with an example:

Imagine that it is desired to create an Entity providing @context terms which are defined in two different JSON-LD @context resources:

- http://example.org/vehicle/v1/vehicle-context.jsonld
- https://schema.org

If a developer wants to reference these two @context resources from a Link header, a wrapper @context can be easily created as follows:

```
{
   "@context": [
       "http://example.org/vehicle/v1/vehicle-context.jsonld",
       "https://schema.org",
       "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
   ]
}
```

As such wrapper @context needs to be referenced from a Link header by using a URI, then it will have to be hosted at some place on the Web. Usually, developers will host @context using popular and simple solutions such as Github or Gitlab pages. As a result, developers will be able to use @context in queries or when using "application/json" as main content type managed by their applications.

It is a **good practice to include the Core @context** in the wrapper @context so it can be used, off-the-shelf, by external JSON-LD processing tools. However, it should be noted this is not necessary for NGSI-LD, as the Core @context is always implicitly included.

Then, using such wrapper @context, (in our example hosted at *https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld*), the developer will be able to issue requests like:

POST /ngsi-ld/v1/entities/

Content-Type: application/json

Content-Length: 200

Link: <https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

```
{
    "id": "urn:ngsi-ld:Vehicle:V1",
    "type": "Vehicle",
    "builtYear": {
        "type": "Property",
        "value": "2014"
    },
    "speed": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    }
}
```

201 Created

Location: /ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:V1

Link: < https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld >; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

GET /ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:V1

Accept: application/ld+json

Link: <https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

200 OK

Content-Type: application/ld+json

```
{
    "id": "urn:ngsi-ld:Vehicle:V1",
    "type": "Vehicle",
    "builtYear": {
        "type": "Property",
        "value": "2014"
    },
    "speed": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    },
    "@context": "https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld"
}
```

Observe that in this case the NGSI-LD Broker is responding with the same wrapper @context in the Link header of the HTTP Response or within the JSON-LD response payload body (when MIME type accepted is "application/ld+json"). However, that could not be always the case, as there could be situations where the NGSI-LD Broker could need to provide a wrapper @context hosted by itself, for instance, when there are inline @context terms or when the Core @context has not been previously included by the wrapper @context (not recommended) provided within developer's requests.

C.9 @context processing clarifications

JSON-LD 1.0 Specification [2] says that "If a term is redefined within a context, all previous rules associated with the previous definition are removed". In addition, it is stated that "Multiple contexts may be combined using an array, which is processed in order".

In contrast to the JSON-LD Specification, the NGSI-LD specification states that the Core @context is protected and has to remain immutable. This essentially means that the Core @context has final precedence and, therefore, is always be processed as the last one in the @context array. For clarity, data providers should place the Core @context in the final position.

From the point of view of Data providers, care has to be taken so that there are no unexpected or undesired term expansions. See the following example:

```
{
    "id": "urn:ngsi-ld:Building:B1",
    "type": "Building",
    "name": {
        "type": "Property",
        "value": "Empire State"
    },
    "openingHours": {
        "type": "Property",
        "value": "Mo-Fr 10am-7pm Sa 10am-22pm Su 10am-21pm"
    },
    "@context": [
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld",
        "https://schema.org"
    ]
}
```

The main caveat of the example above is that the term "name" is defined both by the Core @context and by schema.org and even though schema.org is defined last, the Core @context takes final precedence, the expansion will end up assigning the URI https://uri.etsi.org/ngsi-ld/name to the term "name", which may not be the intent of the Data provider. In these situations, the solution is to prefix the conflicting terms, so that there cannot be any clashing. Therefore, if the intent is to refer to https://schema.org/name throughout, the example above can be modified as shown below:

```
{
    "id": "urn:ngsi-ld:Building:B1",
    "type": "Building",
    "schema:name": {
        "type": "Property",
        "value": "Empire State"
    },
    "openingHours": {
        "type": "Property",
        "value": "Mo-Fr 10am-7pm Sa 10am-22pm Su 10am-21pm"
    },
    "@context": [
        "https://schema.org",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

Note that the Core @context has been placed in the last position of the @context array. NGSI-LD implementations are required to render content following this approach, which has been undertaken in order to maximize compatibility with JSON-LD processing tools. This example works because the "schema:" prefix has already been defined by the schema.org @context.

Annex D (informative): Transformation Algorithms

D.1 Introduction

These algorithms are informative but NGSI-LD implementations should aim at either implementing them as they are described here or devising similar algorithms which take exactly the same input and provides exactly the same output (or an equivalent one as per the JSON-LD specification [2]).

D.2 Algorithm for transforming an NGSI-LD Entity into a JSON-LD document (ALG1)

This algorithm takes as input an NGSI-LD graph which top level node is a particular Entity and returns as output a JSON-LD document which represents all the data associated to the entity. The JSON-LD document (and its associated @context) corresponds to a representation of the Entity in JSON-LD as per the NGSI-LD Information Model.

NOTE: An early implementation of this algorithm can be found at [i.5].

Let:

- **G** be a graph defined as follows:
  - Let **N** be G's top level node.
  - N is an Entity instance of type **T**. Type Name is "AliasT", N's identifier is **I**.
  - N has 0 or more associated Property. Each Property (**Psi**) is defined as follows:
    - Property type identifier is **Pi**.
    - Property Name is "AliasPi".
    - Property Value is **Vi**.
    - Property Value's associated data type is **Di**.
  - N is the subject of 0 or more Relationship. Each Relationship is defined as follows:
    - Relationship type identifier is **Ri**.
    - Relationship nNme is "AliasRi".
    - Relationship target object identifier is **Robji**.
- **O** be a JSON object initialized to the empty object ({}).
- **C** be a JSON-LD @context initialized as described by annex B.

The algorithm should run as follows, provided all the preconditions defined above are satisfied:

1) Add to C a new member <"AliasT", T>.
2) Add to O two new members:
   a) <"id", I>.
   b) <"type", "AliasT">.

3) For each Property Psi (Pi, "AliasP", Vi, Di) associated to N:

   a) Run Algorithm *ALG1.1* taking the following inputs:

      - Ps → Psi.
      - O → O.
      - C → C.

4) For each Relationship Rs (Ri, AliasRi, Robji) associated to N:

   a) Run Algorithm *ALG1.2* taking the following inputs:

      - Rs → Rsi.
      - O → O.
      - C → C.

5) Return (O, C) and end of the algorithm.

D.3 Algorithm for transforming an NGSI-LD Property into JSON-LD (ALG1.1)

Let **Ps** be the Property that has to be transformed. It is defined by (P, "AliasP", V, D), where **P** denotes a Property Type Id, "**AliasP**" is the Property Name, **V** is the Property Value and **D** is the Property Value's data type.

Ps might be associated to extra Properties or Relationships.

Let O be the output JSON-LD object and C the associated JSON-LD context:

1) Execute the following steps:

   a) Add a new member to O with key "AliasP" and value an object structure, let it be named **Op**, and defined as follows:

      - <"type", "Property">.
      - If D is not a native JSON data type add a new member to Op with name "value" and which value has to be an object structure as follows:

        1) <"@type", D>.
        2) <"@value", V>.

      - Else If D is a native JSON data type add a new member to Op as follows:

        1) <"value", V>.

   b) Add a new member to C as follows:

      - <"AliasP", P>.

   c) For each Property associated to Ps (Pss) recursively run the present algorithm (*ALG1.1*) taking the following inputs:

      - Ps → Pss.
      - O → Op.
      - C → C.

d) For each Relationship associated to Ps (Rss) run algorithm *ALG1.2* taking the following inputs:

- Rs → Rss.
- O → Op.
- C → C.

2) Return (O,C) and end of the algorithm.

D.4 Algorithm for transforming an NGSI-LD Relationship into JSON-LD (ALG1.2)

Let **Rs** be the Relationship that has to be transformed. It is defined by (R, "AliasR", Robj), where **R** denotes a Relationship Type Id, "**AliasR**" is the Relationship's Name and **Robj** is the identifier of the target object of the Relationship.

Rs might be associated to extra Properties or Relationships.

Let O be the output JSON-LD object and C the current JSON-LD context:

1) Execute the following statements:

a) Add a new member to O with key "AliasR" and value an object structure, let it be named **Or**, and defined as follows:

- <"object", Robj>.
- <"type", "Relationship">.

b) For each Property associated to Rs (Pss) run the algorithm *ALG1.1* taking the following inputs:

- Ps → Pss.
- O → Or.
- C → C.

c) For each Relationship associated to Rs (Rss) recursively run the present algorithm *ALG1.2* taking the following inputs:

- Rs → Rss.
- O → Or.
- C → C.

2) Return (O,C) and end of the algorithm.

Annex E (informative): RDF-compatible specification of NGSI-LD meta-model

The content of this annex is now in [i.8].

Annex F (informative): Conventions and syntax guidelines

When new concepts or terms are defined they are marked in bold.

EXAMPLE 1: **NGSI-LD Entity** Query Term **observedAt.**

API Parameter names are always in lowercase.

EXAMPLE 2: options.

Entity Types, JSON-LD node types and Data Types are defined using lowercase but with a starting capital letter.

EXAMPLE 3: Vehicle Property Relationship DateTime.

JSON-LD terms are always defined using camel case notation starting with lower case.

EXAMPLE 4: createdAt value unitCode.

When referring to special terms or words, defined previously in the present document or by other referenced specifications, italics format is used.

EXAMPLE 5: *GeoProperty Geometry Second Number*.

When referring to literal strings double quotes are used.

EXAMPLE 6: "application/json" "Subscription".

When referring to the JSON-LD Context the mnemonic text string @context is used as a placeholder.

All the dates and times are given in UTC format.

EXAMPLE 7: 2018-02-09T11:00:00Z.

The measurement units used in the API are those defined by the International System of Units.

EXAMPLE 8: The distance in geo-queries is provided in meters.

When defining application-specific elements or API extensions the same conventions and syntax guidelines should be followed.

Annex G (informative): Change history

Date	Version	Information about changes
February 2017	0.0.1	First draft.
July 2017	0.0.5	Berlin workshop version.
October 2017	0.0.6	After NGSI-LD Information Model Discussions.
November 2017	0.0.7	After Ghent Workshop.
December 2017	0.0.8	After Plenary in Sophia.
January 2018	0.0.10	NGSI-LD. Csource Registration. Csource Subscription.
February 2018	0.0.11	Information Model revised. Terminology revised. Examples revised.
March 2018	0.0.11	Some formal changes in final TB approved draft before first ETSI publication V1.1.1.
March 2018	1.1.1	ETSI Publication as Preliminary Draft.
June 2018	0.1.1 (as WI-009)	First stable version of the NGSI-LD API (pagination, complete query language, csource filters, renamed resource csources/ to csourceRegistrations/).
August 2018	0.4.1	Added batch operations. Minor corrections on query language.
September 2018	0.5.1	Added query by property/relationship. System attribute inclusion.
03rd October 2018	0.6.1	Added Multi-Attribute feature. datasetId.
30th October 2018	0.7.1	Added temporal aspects.
06th November 2018	0.8.1	Added extra error codes. Added lastN parameter. GeoJSON clarifications. MIME types clarifications.
13th November 2018	0.8.2	Added date example. Pagination clarifications.
03rd December 2018	0.9	Added temporal aspects of Csource Registrations. Trailing timezone for DateTime. Reference to schemas and OpenAPI Specification.
14th December 2018	0.9.2	Final review during Sophia-Antipolis Plenary December 2018.
19th December 2018	0.9.3	Technical Officer verifications/checks for ETSI EditHelp registration for Publication.
05th August 2019	1.2.1 (as revised WI-009)	Clarification about @context utilization when expanding / compacting terms. Adddding example. Correcting errata in clause 5.7.2.4. Clarification on Accept header. Context URIs as HTTPs. Clarification on the default @context.
30th August 2019	1.2.1 (as revised WI-009)	Default @context == Core @context. type accepts both URIs and type names. clarifications on the usage of Link header and wrapper @context. link header in 201 and 204. method not allowed clarifications. fixed @context URIs order in examples. context processing rules, made clear, Appendix with example. Examples updated with Core @context always as the last element in the array of @context. Grammar modified to allow prefixed names. Clarified update attributes. Updated UpdateResult structure. Clarified the usage of Temporal Properties. Payload concept unified throughout the document. Added references to schema.org. Clarified behaviour (error code) when remote @context cannot be retrieved.
10 February 2020	1.2.2	Most of the NGSI-LD API and the ETSI ISG CIM information model work referenced here was created with the support of the following European Union Horizon 2020 research projects: No. 732851 (FI-NEXT), No. 723156 (WISE-IoT), No. 732240 (SynchroniCity) and No. 731993 (AutoPilot), No. 814918 (Fed4IoT), No. 779852 (IoTCrawler), No. 731884 (IoF2020).

History

Document history		
V1.1.1	January 2019	Publication
V1.2.1	October 2019	Publication
V1.2.2	February 2020	Publication