ETSI GS CIM 004 V1.1.1 (2018-04)

Context Information Management (CIM); Application Programming Interface (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
DGS/CIM-004-APIprelim

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

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 Specification (preliminary). 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 (preliminary). The Context Information Management API allows users to provide, consume and subscribe to context information in multiple scenarios and involving multiple stakeholders.

The present document is labelled "preliminary" because it will be published widely in order to elicit comment and critique from the ICT community, and their comments will be used to modify and improve the later final API specification. The present document contains two annexes describing a list of pending issues and features that are planned to be addressed in the near future. Accordingly, a feedback process is described in the present document.

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.

1 Scope

The purpose of the present document is the (preliminary) 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 results in the Use Cases [i.1] and Gap Analysis [i.2] work items. The present document is labelled "preliminary" because it will be published widely in order to elicit comment and critique from the user communities and their comments will be used to modify and improve the later final API specification.

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".

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

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

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

[7] IETF RFC 5988: "Web Linking".

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

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

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

[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".

[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".

[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.

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 003: "Context Information Management (CIM); Architecture and Standardization Gaps".

NOTE: Available at https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=51348.

[i.3] OMA OMA-TS-NGSI-Context-Management-V1-0-20100803-C. 03 August 2010: "NGSI Context Management".

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

NOTE: Available at http://www.etsi.org/deliver/etsi_ts/103200_103299/103264/02.01.01_60/ts_103264v020101p.pdf.

[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] J. Frey, K. Müller, S. Hellmann, E. Rahm and M.-E. Vidal, "Evaluation of Metadata Representations in RDF stores", in Semantic Web Journal, 2017.

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

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

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

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions 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 names can be expanded to a fully qualified name in the form of URIs, 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 which 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 a 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 Abbreviations

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

ABNF	Augmented Backus-Naur Form
API	Application Programming Interface
CSV	Comma-Separated Values
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICT	Information and Communication Technology
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
OWL	Ontology Web Language
RDF	Resource Description Format
REST	Representational State Transfer
RFC	Request For Comments
SAREF	Smart Appliance Reference ontology
TB	Technical Body
TLS	Transport Layer Security
TSV	Tab-Separated Values
UCA	Unicode Collation Algorithm
UML	Unified Modelling Language
URI	Uniform Resource Identifier
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 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 I 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 [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 will be extended in later versions with classes defining extra concepts such as mobile vs fixed entities, state properties vs instantaneous vs fixed properties, etc.

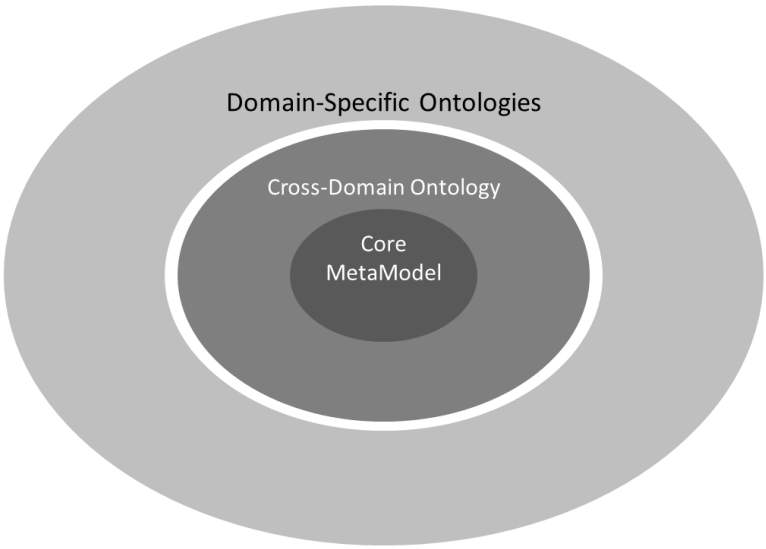

Figure 4.2.1-1: Overview of the NGSI-LD Information Model Structure

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.

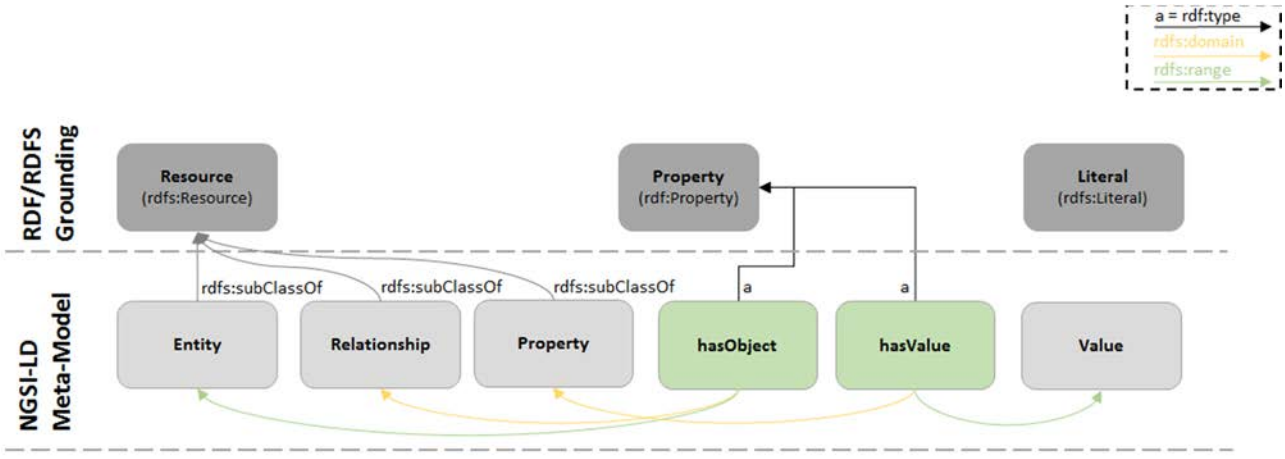

Figure 4.2.2-1: NGSI-LD Core Meta-Model

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 an 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

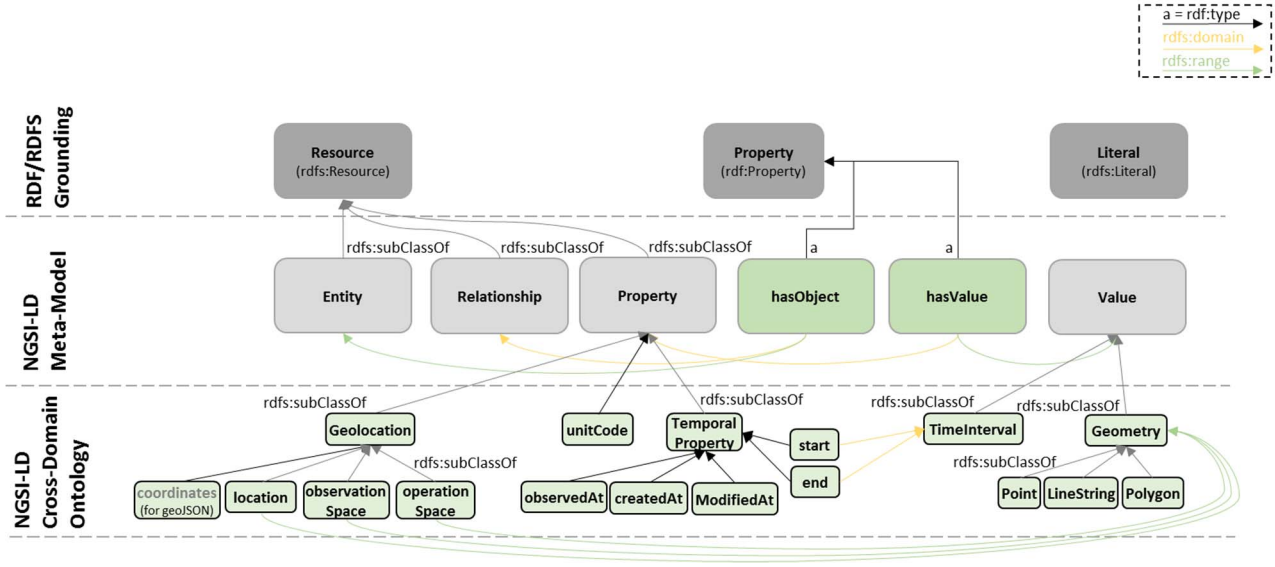

Figure 4.2.3-1: NGSI-LD Core Meta-Model plus the Cross-Domain Ontology

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

- **Location Properties:** Are intended to convey geospatial information and implementations shall support them as defined in clause 4.7.
- **Temporal Properties:** Are intended to convey temporal information and 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 4.8.

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 a 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*.

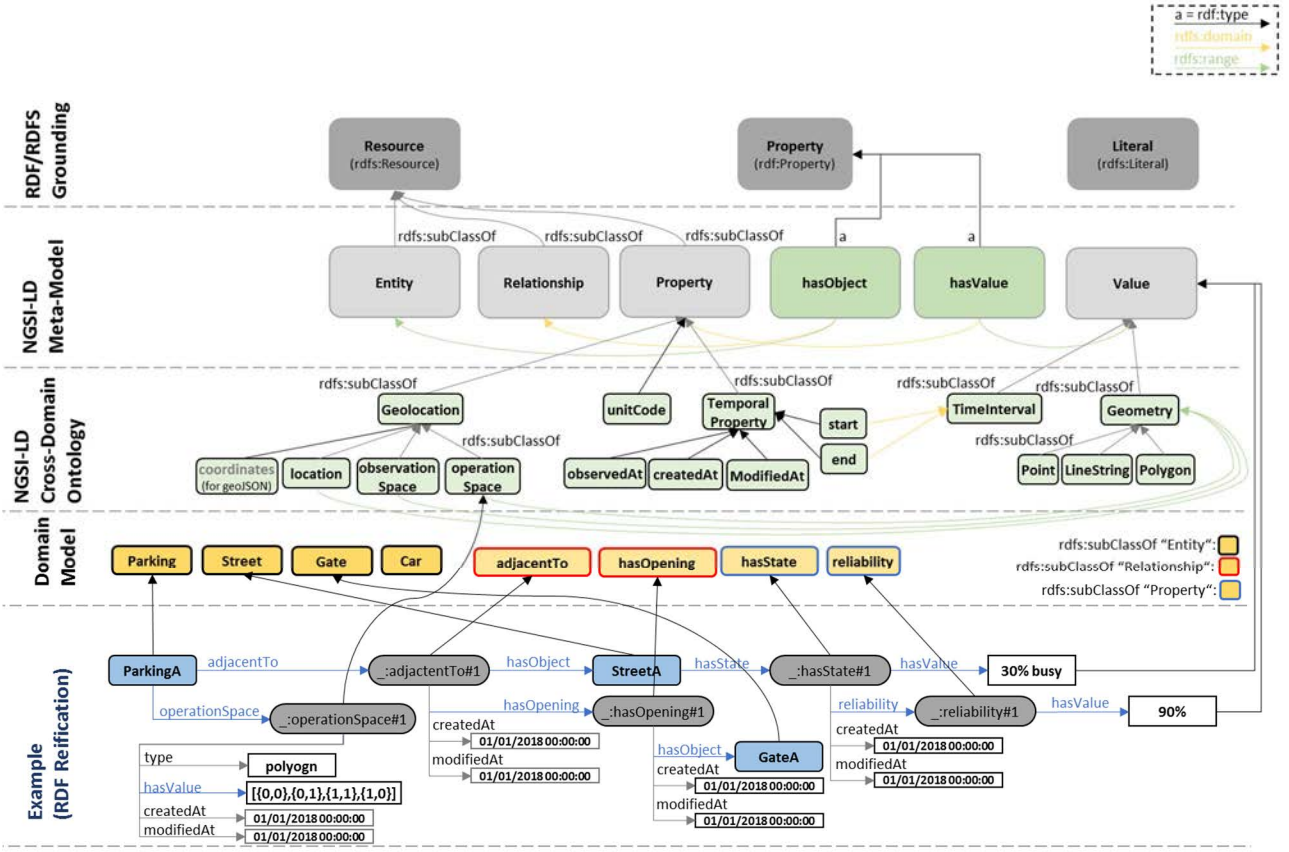

Figure 4.2.4-1: Cross-Domain Ontology and instantiation

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.

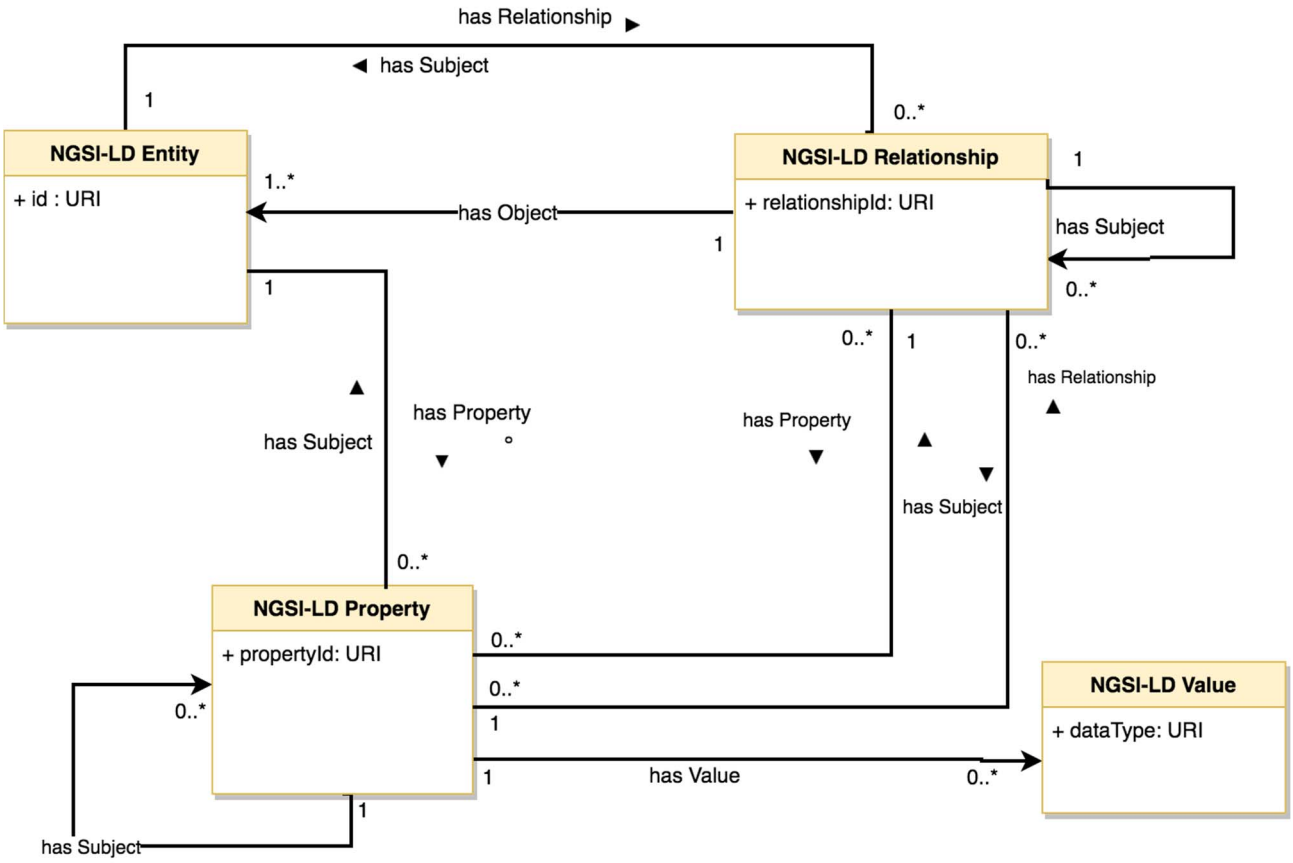

Figure 4.2.5-1: NGSI-LD information model as UML

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 clause 4.3.3 and clause 4.3.4.

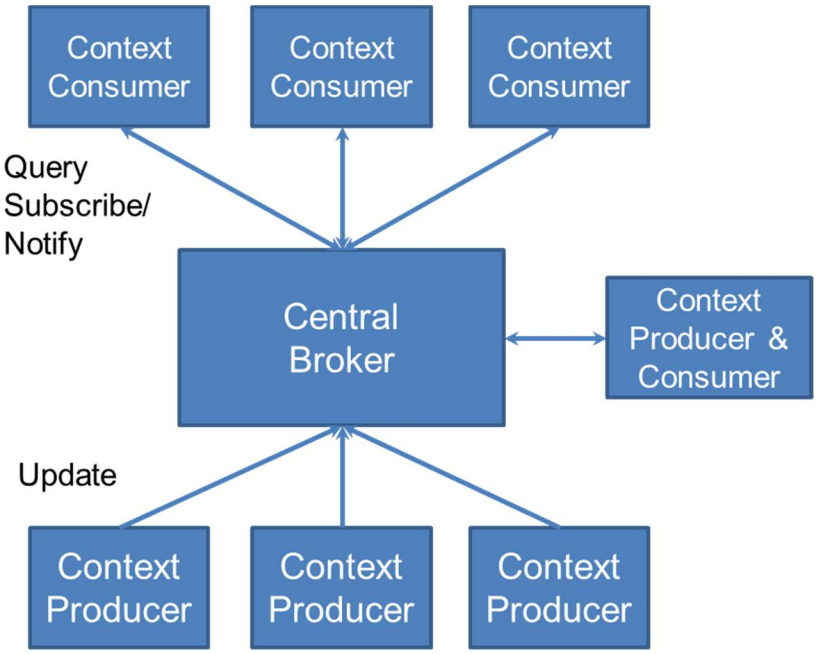

Figure 4.3.2-1: Centralized architecture

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.

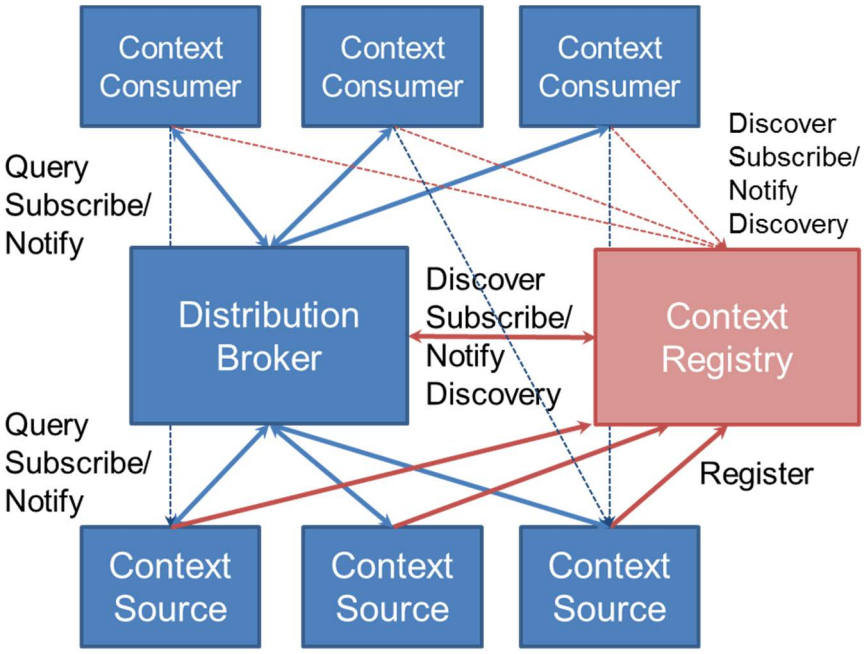

Figure 4.3.3-1: Distributed architecture

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 Server* discovers the domain *Context Brokers* that can provide relevant information, forward the request to these *Brokers* and aggregate the results, so the application gets the result in the same way as in the centralized and distributed cases.

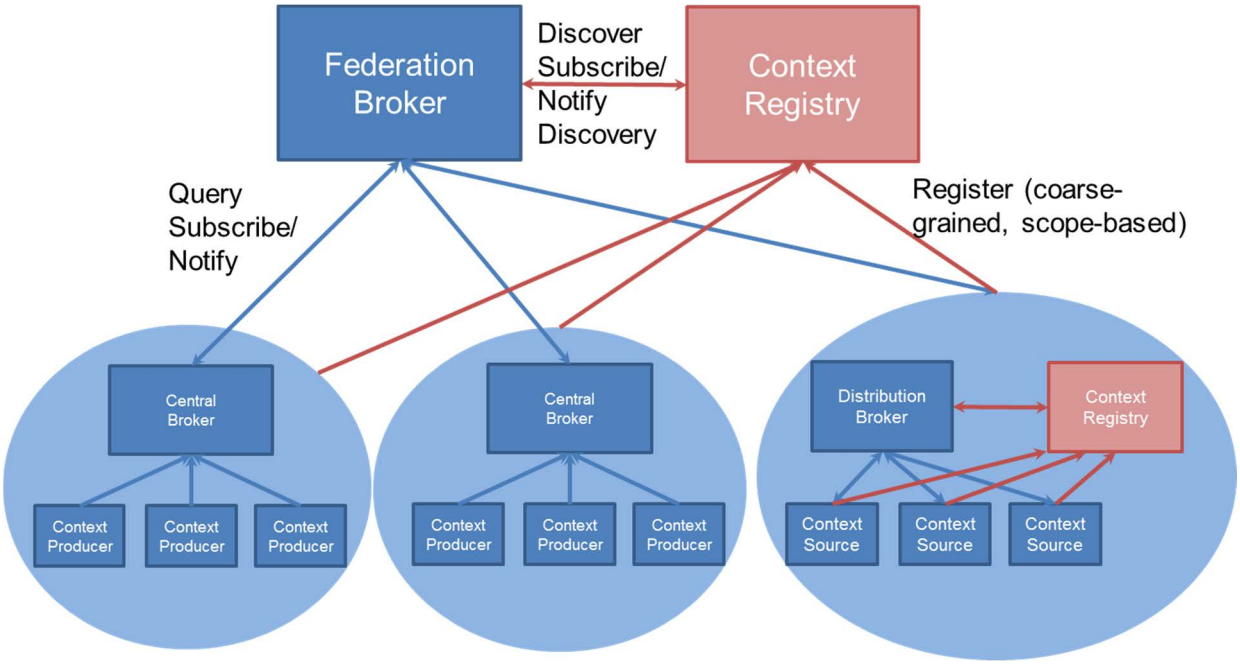

Figure 4.3.4-1: Federated architecture

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

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.

NGSI-LD compliant implementations shall support such Core @context, which shall be implicitly present when processing or generating context information.

The NGSI-LD @context is publicly available at http://xxxxx . 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, a JSON-based format to serialize Linked Data. 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.

Apart from the terms defined by the Core NGSI-LD @context (mandated by annex B), the @context shall contain, at least, 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>.
- 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>.

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

- "id" which value shall be a URI that identifies the Entity.
- "type" which value shall be equal to the Entity Type Name.
- "@context" as mandated by [2], section 5.1. When the context is not available through a Link header (see clause 6.3.5).
- One member for each Property as per the rules stated in clause 4.5.2.
- One member 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) 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*.
- "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 for which 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 for which 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

A NGSI-LD Relationship shall be represented by a member whose key is the Relationship Name (a term) 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*.
- "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 for which 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 for which 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. The simplified representation of Entities shall be supported by implementations and can be selected by Context Consumers through specific request parameters.

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).

4.6 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

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 [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* and *Seconds* components using the format *YYYY-MM-DDThh:mm:ss* 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 and the character ":" is used to separate the time of day 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, *YYYY-MM-DDThh:mm:ss,ssssss.*
  - The timestamp string shall not contain time zone related information. All timestamps shall be interpreted as being 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:ss* 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 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*, which 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. To overcome these issues, optionally, Context Providers can encode the *coordinates* value as a JSON string.

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

For the avoidance of doubt, regular encodings of GeoJSON coordinates (as JSON Array) shall also be accepted by implementations, but Context Producers should take into account the implications in terms of RDF compatibility.

4.8 Temporal properties

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

With the aim of highlighting and encoding those Properties which convey temporal features, NGSI-LD defines a special type of Property named *TemporalProperty*, defined by the NGSI-LD @context as mandated by clause 4.4.

When dealing with NGSI-LD Entities, implementations shall interpret JSON-LD nodes of type *TemporalProperty* just as regular Properties, but with the additional requirement that the Value corresponding to such Property shall be based on *DateTime*, clause 4.6.3.

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

- **observedAt** is defined as the temporal Property at which a certain Property or Relationship became valid or was observed. It is represented as a *DateTime* value. 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. It is represented as a *DateTime* value.
- **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. It is represented as a *DateTime* value.

NOTE: 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.

In Context Source Registrations, the defined temporal Properties are represented as JSON arrays of type *TimeInterval* (see clause 5.2.11) as the Context Source uses it to describe the period of time for which Entity information, i.e. Properties and Relationships, are available. Thus, the temporal Properties it can provide are within the given time intervals. A time interval can be open-ended, i.e. extend up into the present.

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).

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 supported by implementations:

```
Query = QueryTerm *(logicalOp QueryTerm)
QueryTerm = Attribute Operator ComparableValue
QueryTerm =/ Attribute equal CompEqualityValue
QueryTerm =/ Attribute unequal CompEqualityValue
QueryTerm =/ Attribute patternOp RegExp
Attribute = attrName / compoundAttrName / attrPathName
Operator = equal / unequal / greater / greaterEq / less / lessEq
ComparableValue = Number / quotedStr / dateTime / date / time
OtherValue = false / true / null
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                                               ; ~=
dots = %x2E %x2E                                                    ; ..
attrNameChar = %x21 / %x23 / %x24                                   ; ! / # / $
attrNameChar =/ DIGIT / ALPHA
attrNameChar =/ %x5F / %x7E                                         ; _ / ~
attrName = 1*attrNameChar
attrPathName = attrName *(%x2E attrName).                           ; attrName *(. attrName)
compoundAttrName = attrName *(%x5B attrName %x5D)                   ; . attrName *([ attrName ])
quotedStr = String / %x27 *char %x27                                ; '*char'
andOp = %x3B                                                        ; ;
logicalOp = andOp
```

- *DIGIT* and *ALPHA* are defined by [12].
- *Number* shall be a number as mandated by the JSON Specification, following the ABNF Grammar, production rule named *number*, section 6 of [6].
- *String* shall be a text string as mandated by the JSON Specification, following the ABNF Grammar, production rule named *String*, section 7 of [6].
- *char* shall be a character as mandated by the JSON Specification, ABNF Grammar, production rule named *char*, section 7 of [6].
- *false* shall be conformant with the JSON ABNF Grammar, production rule named *false*, section 3 of [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 [6]. It is intended to represent the Boolean value corresponding to "true".
- *null* shall be conformant with the JSON ABNF Grammar, production rule named *null*, section 3 of [6]. It is intended to represent a JSON value of type ***null***.
- *RegExp* shall be a regular expression as mandated by [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 [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 operator (production rule named *Operator*).
- a value (production rule named *Value*.

EXAMPLE 1: temperature==20.

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

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.

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 syntax of an attribute path is defined by the production rule *Attribute*, as a list of names. Such 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 the target element is a Relationship, the **target object** is defined as the object associated (represented as a URI) to such Relationship.

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 [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 [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 *lower*). 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 *lowerEq*). 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.

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 [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 ([8], section 1.4), except *GeometryCollection*.
- A coordinates (parameter name **coordinates**) element which shall represent the coordinates of the reference geometry as mandated by [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).

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. In case only the latest Value is stored the temporal query simply acts as a filter for this value.

The following grammar defines the syntax for the temporal query that shall be supported:

```
temprel = 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 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 property *observedAt*.

EXAMPLE 1: temprel=before

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

EXAMPLE 2: temprel=between

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

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

timeproperty=modifiedAt

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

?type=Vehicle&temprel=between&time=2017-12-13T14:20:00&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 'timestamp' as default) has to be before the time specified in 'time'.
- **after** relationship (production rule named 'afterRel'). For a temporal property to match, the value of the specified temporal property (or 'timestamp' as default) has to be after the time specified in 'time'.
- **between** relationship (production rule named 'betweenRel'). For a temporal property to match, the value of the specified temporal property (or 'timestamp' as default) has to be after the time specified in 'time' and before the time specified in 'endtime'.

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

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.

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 involving the referred NGSI-LD elements, implementations (unless they are asked to be omitted by the Context Consumer) shall generate them as part of their representation.

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"), it shall be included a proper @context 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 a 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	string	Entity Type Name	1	Entity Type
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 [6]	See NGSI-LD Value definition at 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
<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
<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
<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. At least one of the JSON members below shall be present.

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.

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 [11]	0..1	A regular expression which denotes a pattern that shall be matched by the provided or subscribed Entities
type	string	Entity Type Name as short-hand string. See clause 4.6.2	1	Entity Type Name

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.

None of the members described permit 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.

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
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	1	Describes the Entities, Properties and Relationships for which the Context Source may be able to provide information
timestamp	TimeInterval[]	See data type definition in clause 5.2.11	0..1	Time intervals for which the Context Source may be able to provide information. If not provided the assumption is that only the latest available values can be provided
location	GeoJSON Geometry as mandated by 4.7		0..1	Location for which the Context Source may be able to provide information
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

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	1	Describes the entities for which the CSource may be able to provide information.
properties	string []	Property Name as short-hand string	0..1	Describes the Properties that the CSource may be able to provide for the Entities described in the *entities* member.
relationships	string []	Relationship Name as short-hand string	0..1	Describes the Relationships that the CSource may be able to provide for the Entities described in the *entities* member.

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.

None of the members described permit 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.

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	1	Entities subscribed
watchedAttributes	string[]	Attribute Name as short-hand string. if *timeInterval* is present it shall not appear (0 cardinality)	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	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
notification	NotificationParams	See data type definition on clause 5.2.14	0..1	Notification details
expires	string	*DateTime* (see clause 4.6.3)	0..1	Expiration date for the subscription
status	string	Allowed values: "active" "inactive" "failed" "expired"	0..1	Provided by the system when querying the details of a subscription
throttling	Number	Greater than 0	0..1	Minimal period of time in seconds which shall elapse between two consecutive notifications

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 [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.)

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.	0..1	Entity Attribute Names (Properties or Relationships) to be included in the notification payload. 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 end point details

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 (unless they are asked to be omitted by the Context Subscriber or Consumer) 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 (JSON or JSON-LD)

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] 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 [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
http://uri.etsi.org/ngsi-ld/errors/InvalidRequest	The request associated to the operation is syntactically invalid or includes wrong content
http://uri.etsi.org/ngsi-ld/errors/BadRequestData	The request includes input data which does not meet the requirements of the operation
http://uri.etsi.org/ngsi-ld/errors/AlreadyExists	The referred element already exists
http://uri.etsi.org/ngsi-ld/errors/OperationNotSupported	The operation is not supported
http://uri.etsi.org/ngsi-ld/errors/ResourceNotFound	The referred resource has not been found
http://uri.etsi.org/ngsi-ld/errors/InternalError	There has been an error during the operation execution

5.5.3 Error payloads

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

- **type:** Error type as per clause 5.4.1.
- **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 [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 input payload 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 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.

5.5.5 Default @context assignment

If an input JSON document provided by an API client, does not include a @context and there is no other mechanism available to determine it, then the implementation shall assign a default @context to such JSON document. A 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 end point 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.

5.5.7 Term to URI expansion

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 operations, NGSI-LD systems shall expand terms to URIs, in order to obtain fully qualified names.

The term to URI expansion 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 shall be performed using a default @context (clause 5.5.5).

If a term cannot be expanded as per the supplied @context then it should be expanded using a default JSON-LD @context.

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 [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, which value is different than *null*, contained by the target, the target member value is replaced by value given in the Fragment.
- For each member of the Fragment, which value is *null*, contained by the target, the target member is removed.

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.

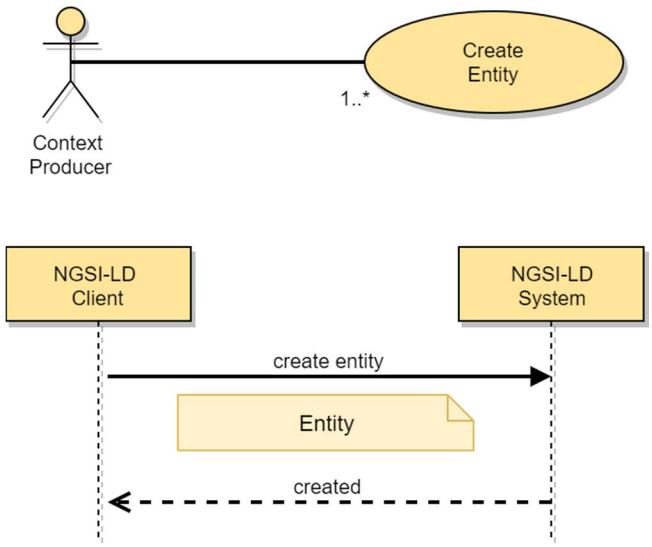

Figure 5.6.1.2-1: Create entity use case

5.6.1.3 Input data

A JSON-LD document representing a 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 end point 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.

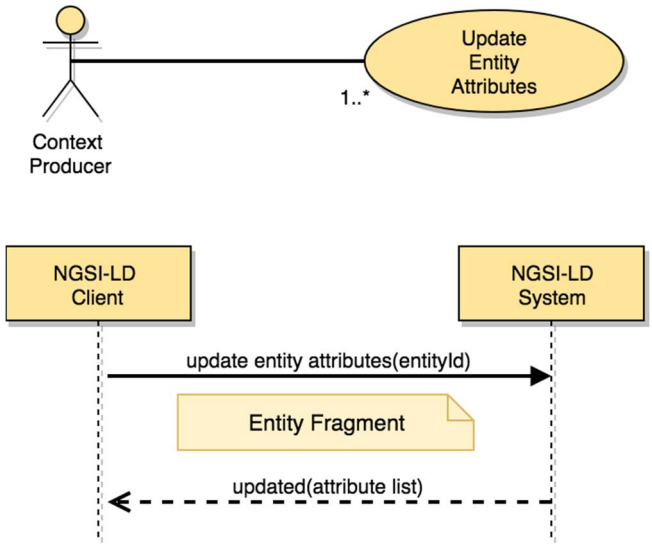

Figure 5.6.2.2-1: Update entity Attributes use case

5.6.2.3 Input data

- A URI representing the id of the Entity to be updated (target Entity).
- A JSON-LD document representing a 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 end point 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 by the Fragment, if the target Entity includes a matching one (taking into account term expansion rules as mandated by clause 5.5.7) then replace it by the one included by the Fragment. Otherwise ignore it.

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 a 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.

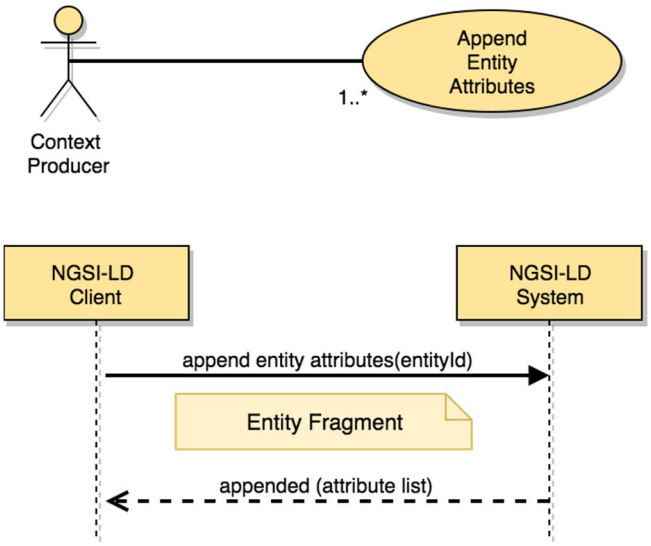

Figure 5.6.3.2-1: Append Entity Attributes use case

5.6.3.3 Input data

- A URI representing the id of the E to be modified (target Entity).
- A JSON-LD document representing a 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 end point 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 (taking into account 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 (taking into account term expansion rules as mandated by clause 5.5.7):
    - 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 a 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.

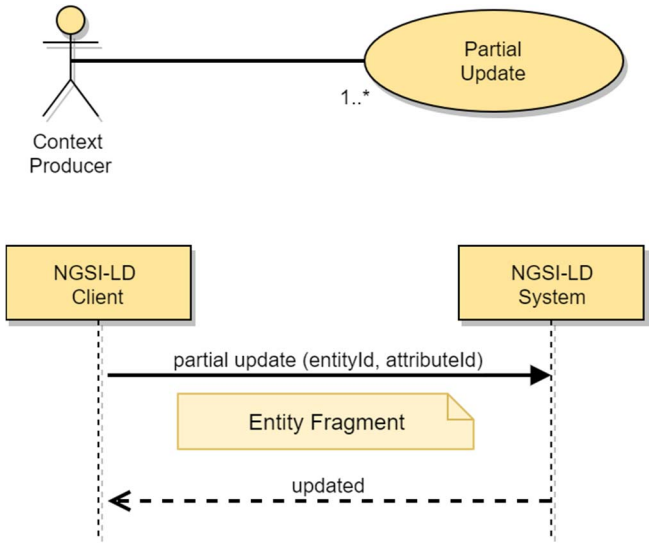

Figure 5.6.4.2-1: Partial Attribute update use case

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 a 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 end point 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 a 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.

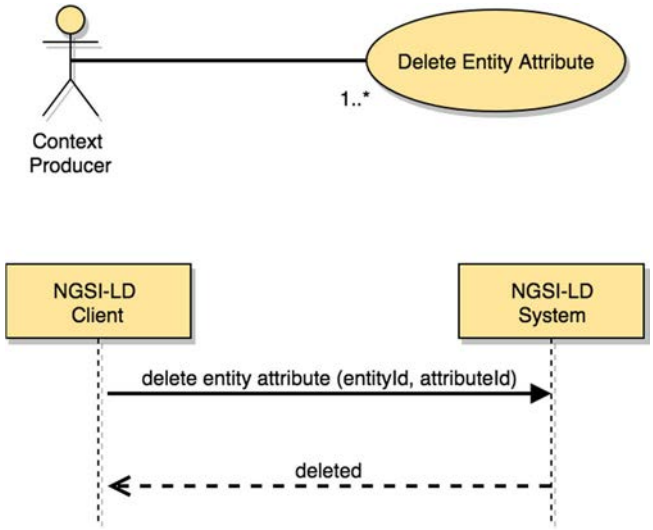

Figure 5.6.5.2-1: Delete Entity Attribute use case

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 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 end point 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 5.4.6 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.
- Remove the target Attribute from the target Entity.

5.6.5.5 Output data

None.

5.6.6 Delete Entity

5.6.6.1 Description

This operation allows deleting a 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.

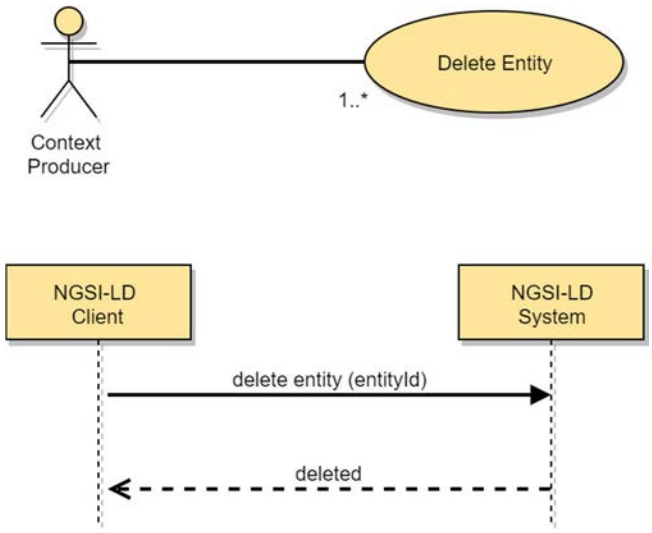

Figure 5.6.6.2-1: Delete Entity use case

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 end point 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.

5.7 Context Information Consumption

5.7.1 Retrieve Entity

5.7.1.1 Description

This operation allows retrieving a 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.

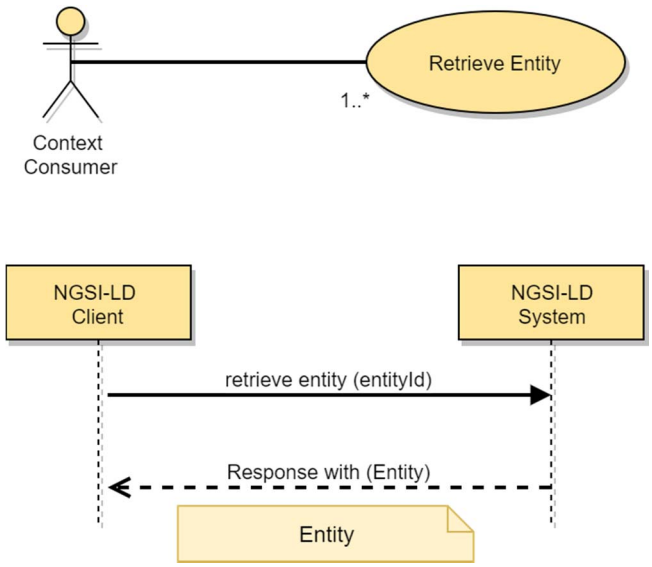

Figure 5.7.1.2-1: Retrieve Entity use case

5.7.1.3 Input data

- Entity Id (URI) of the Entity to be retrieved (target Entity).
- List of Entity Attributes (Properties or Relationships) to be retrieved (as names).
- 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 end point 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 a 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.

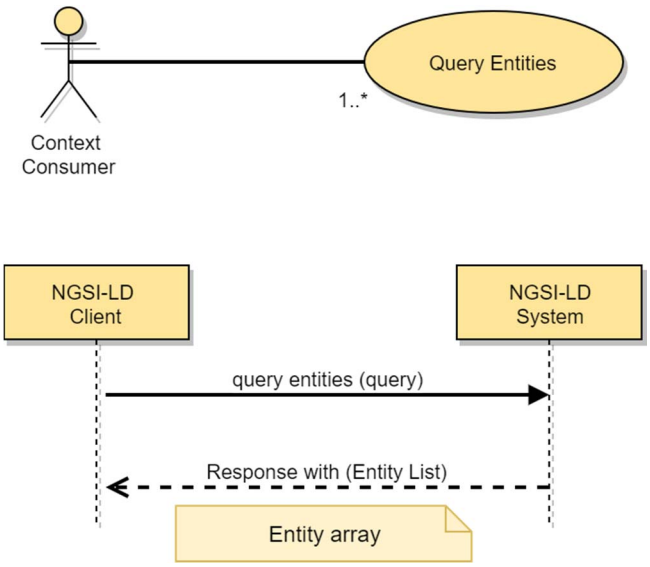

Figure 5.7.2.2-1: Query entities use case

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 (mandatory).
- 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 mandated by clause 4.9.
- An NGSI-LD geoquery (optional) as mandated by clause 4.10.
- An NGSI-LD temporal query (optional) as mandated by clause 4.11.

5.7.2.4 Behaviour

- If an Entity type or Entity id 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.

- 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 present, the temporal restrictions imposed by the temporal query are met (as mandated by clause 4.11).

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.

5.8 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.

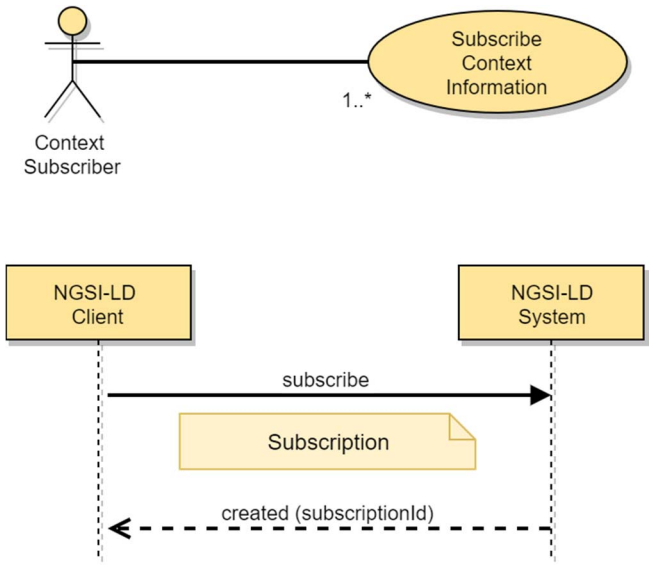

Figure 5.8.1.2-1: Create subscription use case

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 end point already knows about this subscription, because 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 subscription is not expired and no status is provided then the initial status of the subscription shall be set to "active".
  - 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 term to URI expansion) pertaining to the subscribed entities and conveyed through the "attributes" member. The watched Attributes are those that trigger a new notification when they are changed. A non-present *watchedAttributes* member or a *watchedAttributes* member of length 0 means that all Attributes shall be watched.
  - If the document does not include the "status" member, the initial status of the subscription will be "active", provided it has not expired yet.
- 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.

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 a 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.

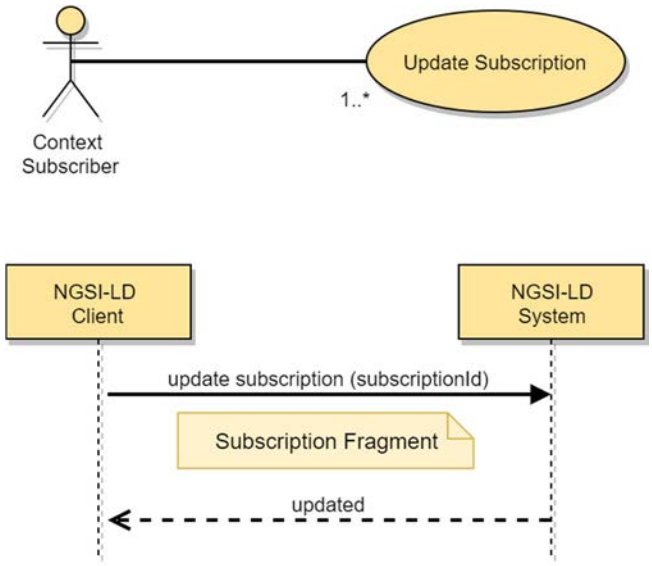

Figure 5.8.2.2-1: Update subscription use case

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.
- 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.

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.

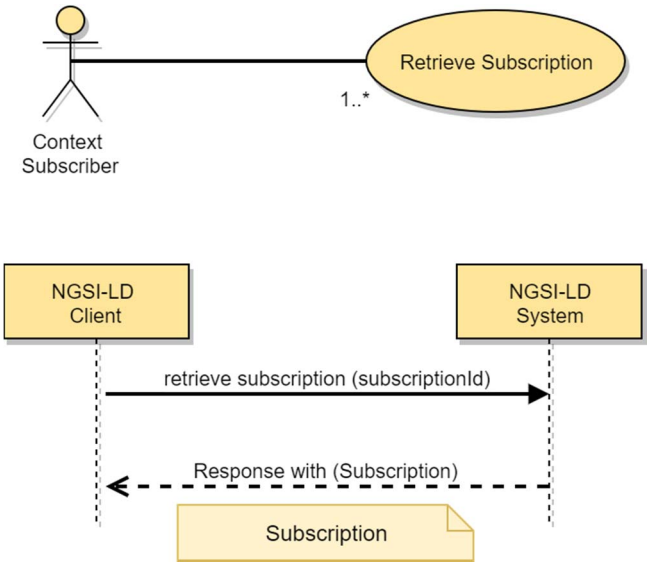

Figure 5.8.3.2-1: Retrieve subscription use case

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

NOTE: This functionality will be reviewed in the next release, as it is currently specified in its most simple fashion.

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.

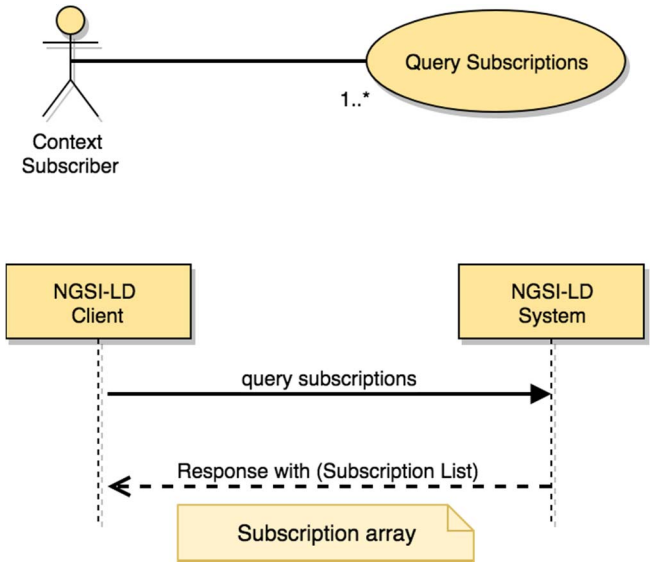

Figure 5.8.4.2-1: Query subscriptions use case

5.8.4.3 Input data

None.

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.

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.

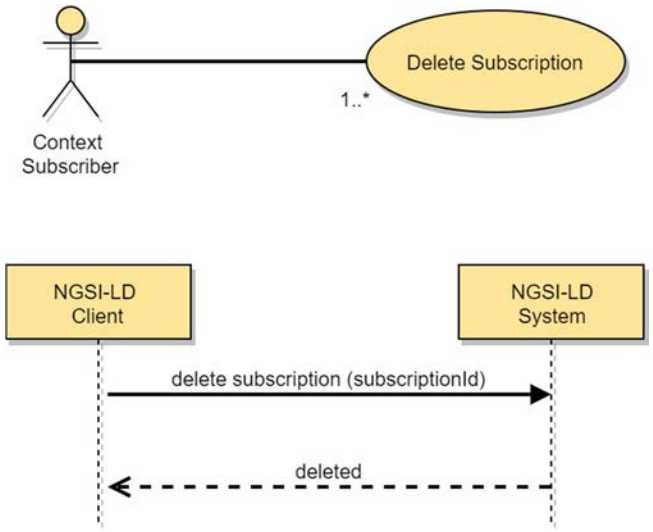

Figure 5.8.5.2-1: Delete subscription use case

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 is different than "inactive" or "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 the query and geoquery conditions. If query or geoquery are not defined then all subscribed entities that changed 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).
  - 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 the current timestamp.
  - If the response to the notification request is 200 OK then implementations shall:
    - Update "notification.lastSuccess" with the current timestamp.
  - If the response to the notification request is different than 200 OK then implementations shall:
    - Update "notification.lastFailure" with the current timestamp.
    - Update the subscription "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.

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.

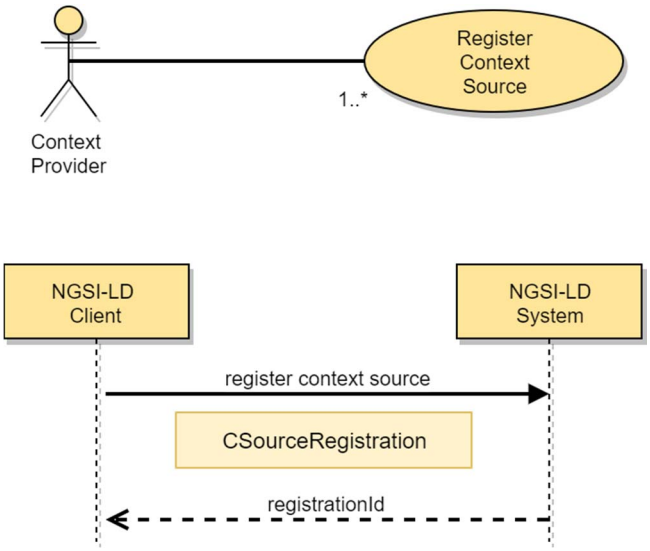

Figure 5.9.2.2-1: Register context source use case

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 'type' is 'null', not defined or does not represent a valid URI equivalent to the reserved type "ContextSourceRegistration", an error of type "Bad Request Data" shall be raised.
- If the property 'url' is 'null', not defined or does not represent a valid URI, an error of type "Bad Request Data" shall be raised.
- If the property 'information' is 'null', not defined, or is not a valid JSON array consisting only of elements of type *RegistrationInfo* as specified in clause 5.2.10, an error of type "BadRequestData" shall be raised.
- If property 'timestamp' is contained in the Context Source Registration, its value has to be a valid JSON array consisting only of elements of type TimeInterval as specified in clause 5.2.11. Otherwise an error of type "BadRequestData" shall be raised.
- If property 'location' is contained in the Context Source Registration, its value has to be a valid GeoJSON Geometry as specified in clause 4.7. Otherwise an error of type "BadRequestData" shall be raised.
- If the property 'expires' is 'null' or 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.

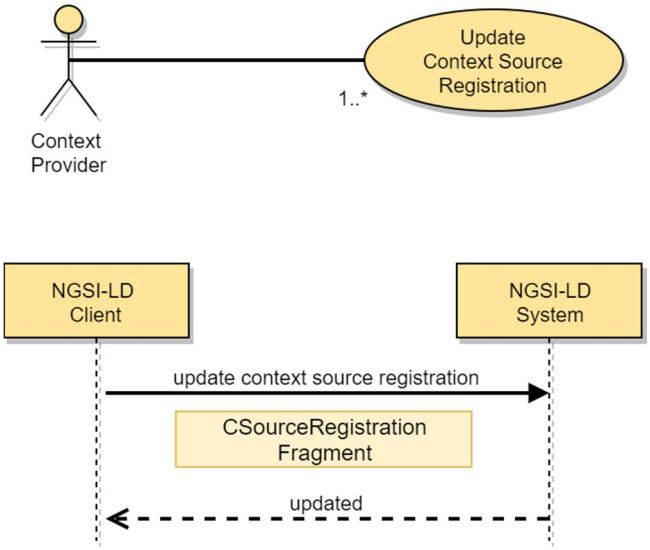

Figure 5.9.3.2-1: Update context source registration use case

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.
- 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.

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.

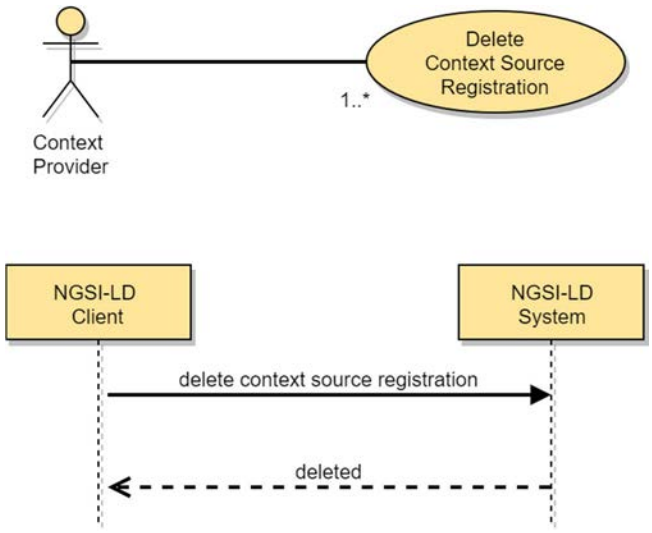

Figure 5.9.4.2-1: Delete context source registration use case

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 end point 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.

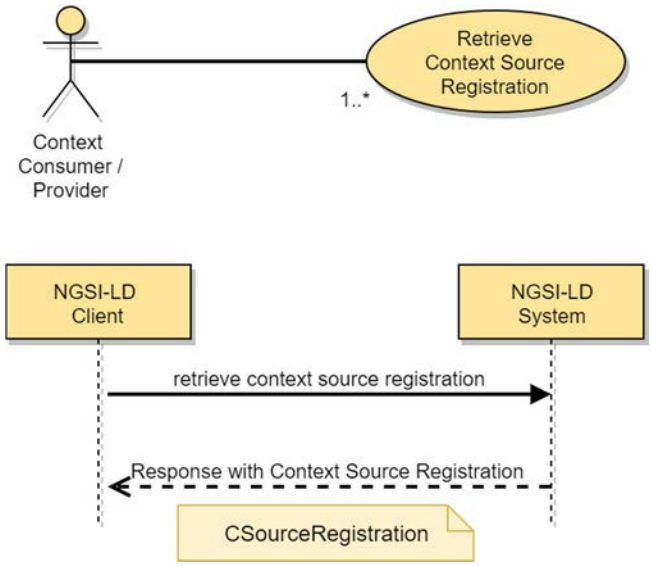

Figure 5.10.1.2-1: Retrieve context source registration use case

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 end point 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.

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.

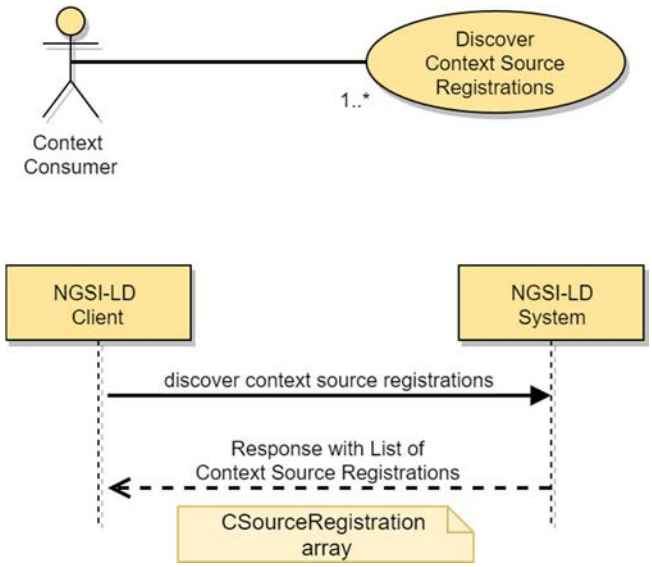

Figure 5.10.2.2-1: Discover context source registrations use case

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 (mandatory).
- A list (one or more) of Entity identifiers (optional).
- A list (one or more) of Attribute identifiers (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.

5.10.2.4 Behaviour

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- If an Entity type or Entity id 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, 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 the following conditions:
  - The entity specification in the query consisting of a combination of entity type and entity id/entity id pattern (if present) matches an *EntityInfo* specified in a *RegistrationInfo* of the context source registration. This matching is further described in clause 5.12.
  - If present, at least one attribute identifier specified in the query matches one property or Relationship in the same *RegistrationInfo* element that matched the entity specification above. If no Properties or Relationships are specified in the *RegistrationInfo*, it is automatically considered a match. 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 present, the temporal query is matched against the 'timeproperty' identified in the temporal query. If no 'timeproperty' is specified in the temporal query, the default property is 'timestamp'. The semantics of the match is that the 'time' in the case of the 'before' and 'after' relation is contained in or be an endpoint of a time period included in the temporal property of the context source registration. 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 a time period included in the temporal property of the context source registration.
    If the 'timeproperty' is not present in a Context Source Registration, the assumption is that the Context Source can provide the latest value and it matches if the temporal query is based on the 'after' relation.

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 discovery 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.

5.11.2 Create Context Source Registration Subscription

5.11.2.1 Description

This operation allows creating a new Context Source discovery subscription.

5.11.2.2 Use case diagram

A Context Source subscriber can subscribe to a new context source discovery as shown in figure 5.11.2.2-1.

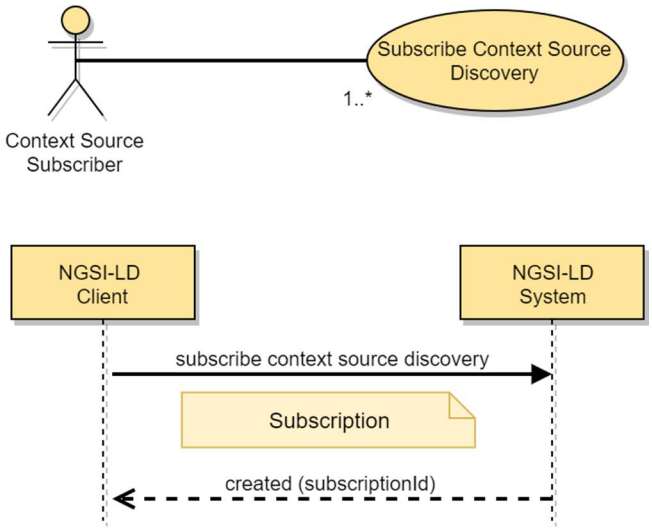

Figure 5.11.2.2-1: Subscribe context source discovery use case

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 discovery 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 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 discovery subscription

5.11.3.1 Description

This operation allows updating an existing context source discovery subscription.

5.11.3.2 Use case diagram

A context source subscriber can update a context source discovery subscription as shown in figure 5.11.3.2-1.

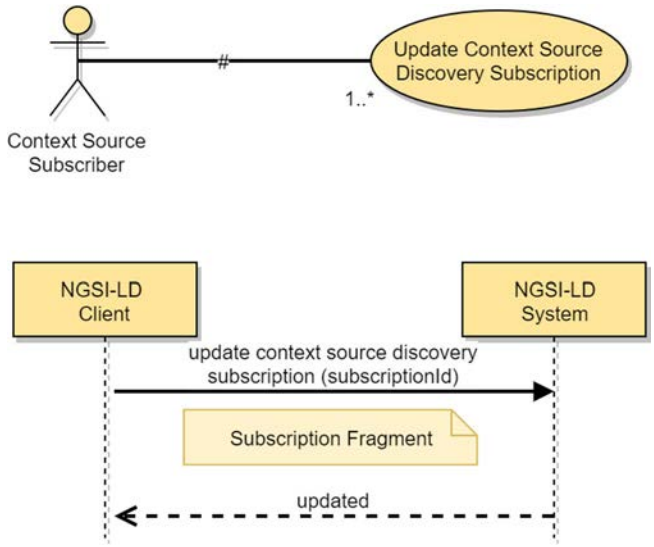

Figure 5.11.3.2-1: Update context source discovery subscription use case

5.11.3.3 Input data

- Subscription identifier (URI), the target Context Source discovery 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.
- 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 discovery subscription

5.11.4.1 Description

This operation allows retrieving an existing Context Source discovery subscription.

5.11.4.2 Use case diagram

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

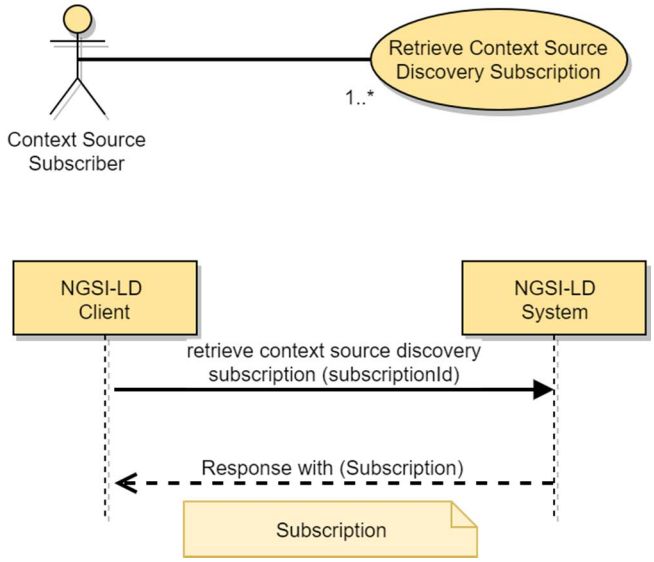

Figure 5.11.4.2-1: Retrieve context source discovery subscription use case

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 discovery 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 Discovery subscriptions

5.11.5.1 Description

This operation allows listing existing Context Source discovery subscriptions.

5.11.5.2 Use case diagram

A context source subscriber can list all existing context source discovery subscriptions as shown in figure 5.11.5.2-1.

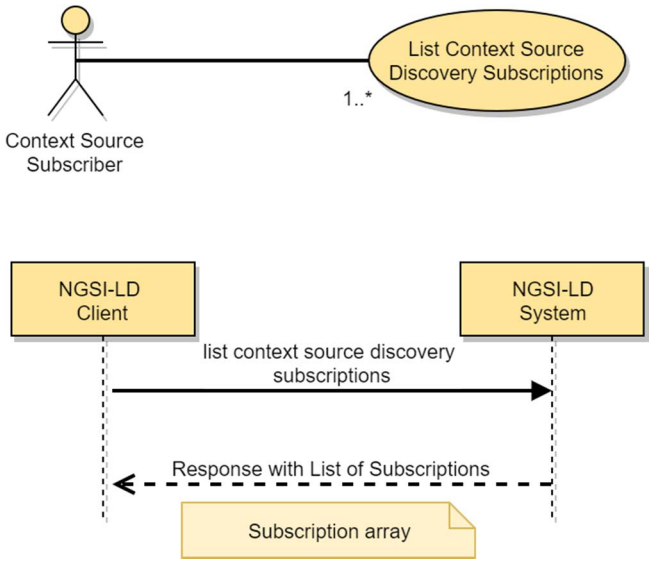

Figure 5.11.5.2-1: Retrieve context source discovery subscription use case

5.11.5.3 Input data

- Maximum number of subscriptions to be retrieved.

5.11.5.4 Behaviour

- The NGSI-LD system shall list all the existing context source discovery subscriptions up to the limit specified as input data. If no limit is specified the number of subscriptions retrieved may depend on the implementation.

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 discovery subscription

5.11.6.1 Description

This operation allows deleting an existing context source discovery subscription.

5.11.6.2 Use case diagram

A context source subscriber can delete a context source discovery subscription as shown in figure 5.11.6.2-1.

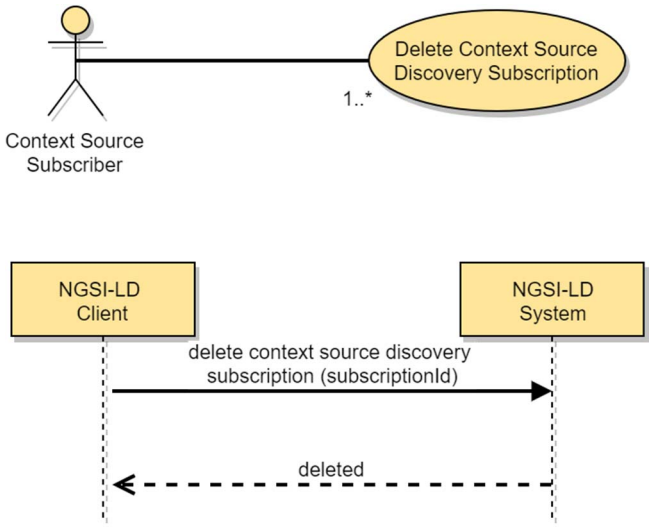

Figure 5.11.6.2-1: Retrieve context source discovery subscription use case

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 discovery 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 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 Entity specification information in the query for Context Source Registrations are the list of Entity Types, the list of Entity identifiers (if present), the id pattern (if present) and the list of Attribute identifiers (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 *EntityInfo* specification refers to the relevant elements for matching, i.e. Entity Types, Entity identifiers, id pattern and Attribute identifiers. Except for the mandatory Entity Types, all other elements are optional.

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

- The Entity Types, Entity identifiers and id pattern match at least one of the *EntityInfo* elements (see below).
- The Attribute identifiers match the combination of properties and relationships specified in the *RegistrationInfo* (see below).

An *EntityInfo* 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 identifiers match the combination of Properties and Relationships if one of the following conditions hold:

- No Attribute identifiers 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 identifier 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, input and output payloads.

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.

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 and 6.13 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.

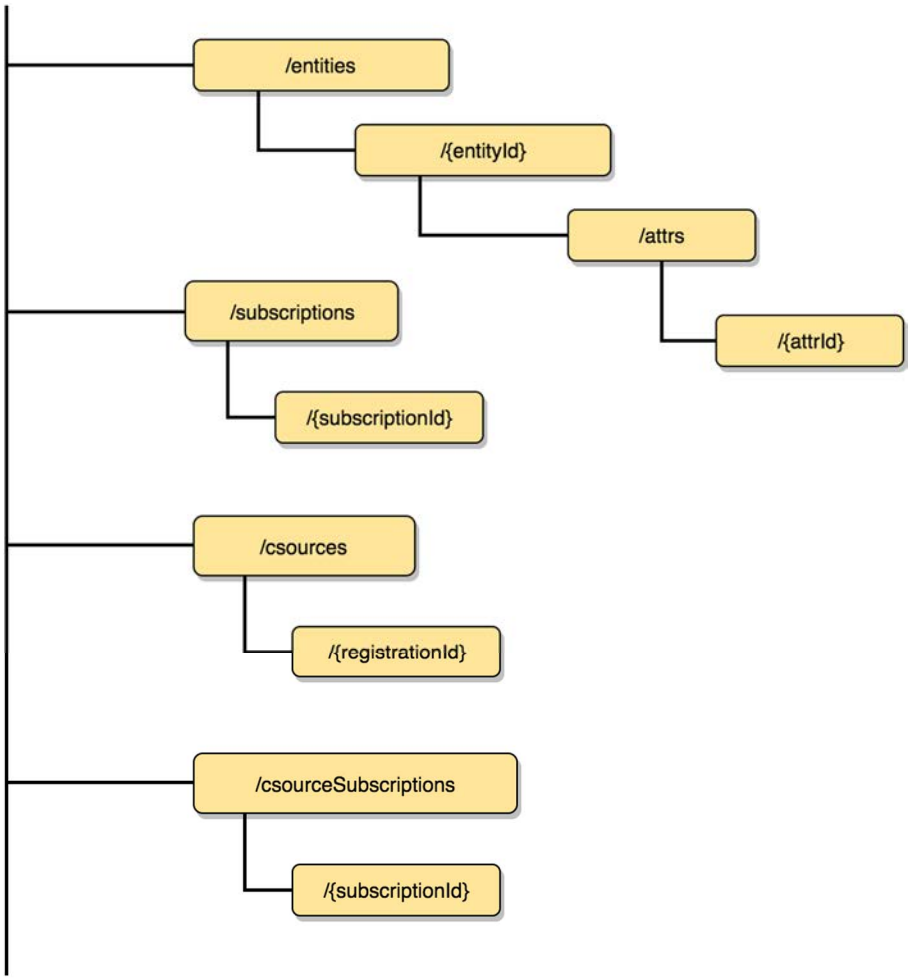

Figure 6.2-1: Resource URI structure of the NGSI-LD API

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	/csources	POST	Csource registration creation
		GET	Discover Csource registrations
Context source registration by Id	/csources/{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

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 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
http://uri.etsi.org/ngsi-ld/errors/InvalidRequest	400
http://uri.etsi.org/ngsi-ld/errors/BadRequestData	400
http://uri.etsi.org/ngsi-ld/errors/AlreadyExists	409
http://uri.etsi.org/ngsi-ld/errors/OperationNotSupported	422
http://uri.etsi.org/ngsi-ld/errors/ResourceNotFound	404
http://uri.etsi.org/ngsi-ld/errors/InternalError	500

In addition, implementations shall support specific errors of the HTTP binding, particularly "Method Not Allowed" (405) which shall be raised when a client invokes a wrong HTTP verb over a resource.

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: 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 input payload.

For PATCH HTTP requests "application/merge-patch+json" is allowed as Content-Type, as mandated by [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 be "application/json", "application/ld+json" or "*/*". A non-present accept header is also allowed. In that particular case "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.

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.
- 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 input payload itself. If no @context can be obtained from the input payload, then an HTTP error response of type *BadRequestData* shall be raised.

6.3.6 HTTP response common requirements

Implementations shall honour the Accept header provided by HTTP requests as follows:

- If the Accept header contains "application/json" but not "application/ld+json" then the response's Content-Type shall be "application/json" and such response shall include a Link to the associated JSON-LD @context as mandated by [2], clause 6.8.

- If the Accept header contains "application/ld+json", then the output payload provided by the HTTP response shall include a JSON-LD @context.

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).

6.4 Resource: entities

6.4.1 Description

This resource represents a collection of entities known to a 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 input payload. 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.

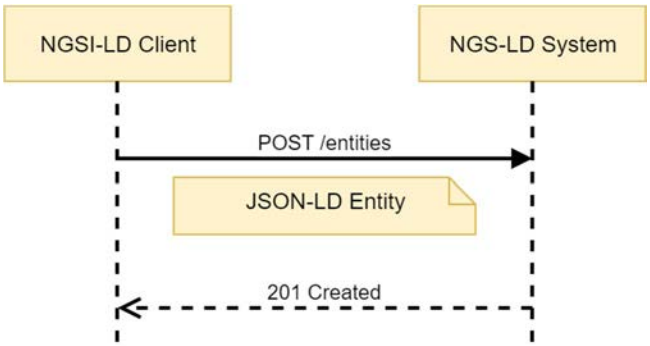

Figure 6.4.3.1-1: Create Entity interaction

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 output payload. Figure 6.4.3.2-1 shows the query entities interaction.

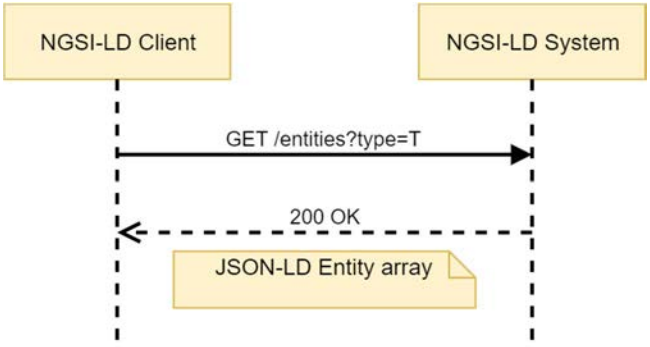

Figure 6.4.3.2-1: Query Entities interaction

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 type names	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
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
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

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 a 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.

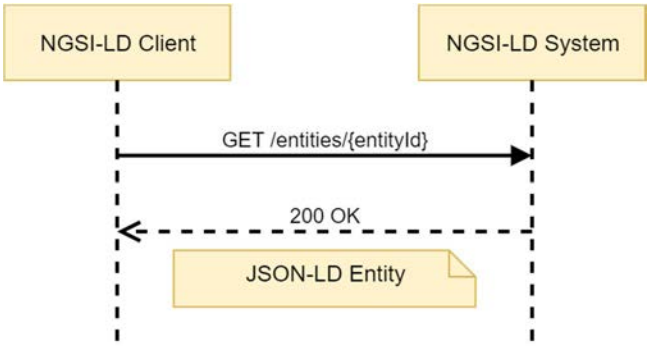

Figure 6.5.3.1-1: Retrieve Entity interaction

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.

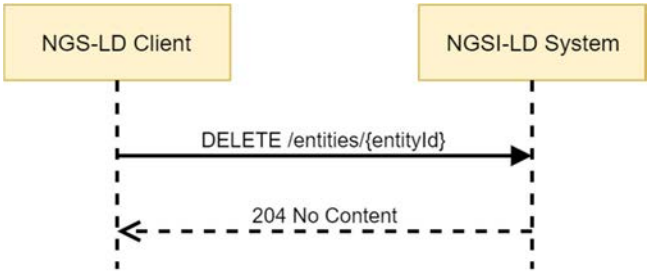

Figure 6.5.3.2-1: Delete Entity interaction

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 a 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 input payload. 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.

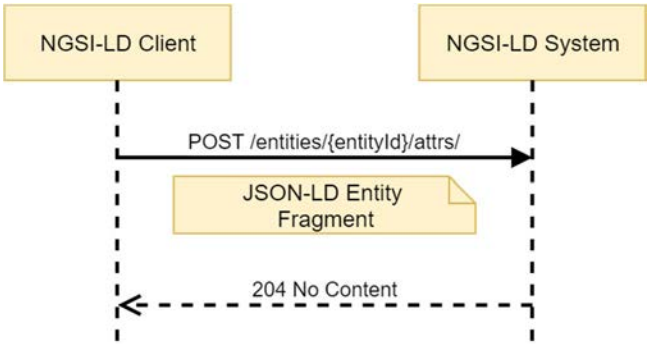

Figure 6.6.3.1-1: Append Entity Attributes interaction

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 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 appended shall be contained in the HTTP request input payload. 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.

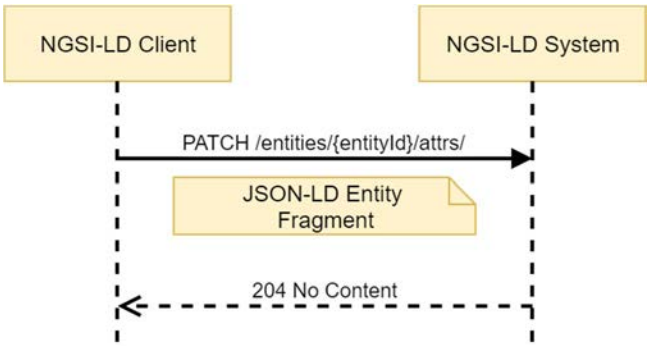

Figure 6.6.3.2-1: Update Entity Attributes interaction

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 were 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 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 a 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 input payload. 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.

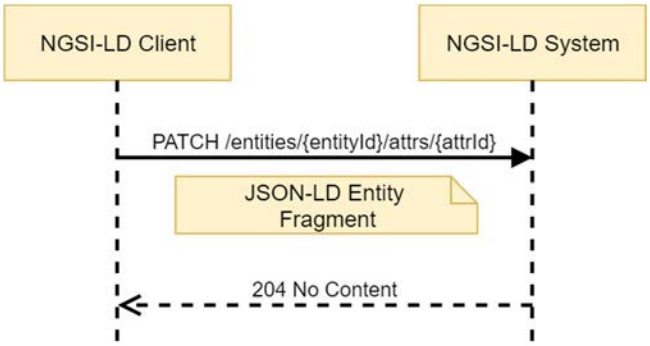

Figure 6.7.3.1-1: Partial Attribute Update interaction

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 and table 6.7.3.2-1 describes the request body and possible responses.

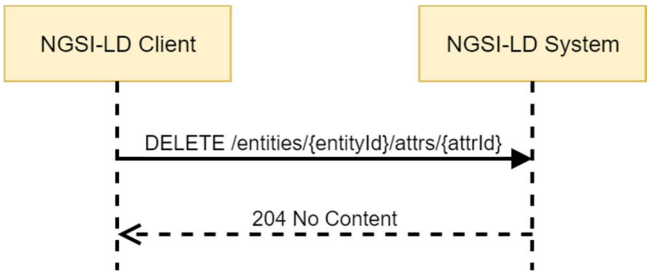

Figure 6.7.3.2-1: Delete Entity Attribute interaction

Table 6.7.3.2-1: 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: csources

6.8.1 Description

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

6.8.2 Resource definition

Resource URI:

- /csources

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 input payload. 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.

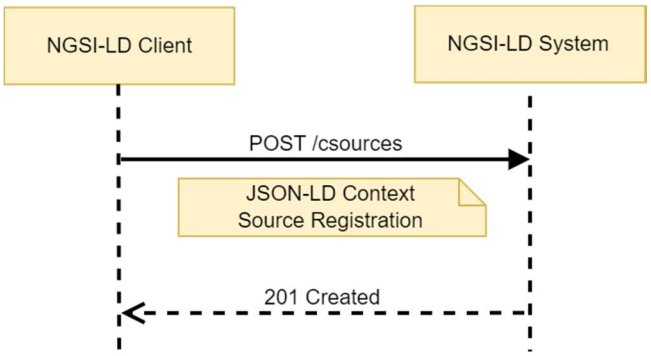

Figure 6.8.3.1-1: Register Context Source interaction

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 output payload. Figure 6.8.3.2-1 shows the Query Context Source Registrations interaction.

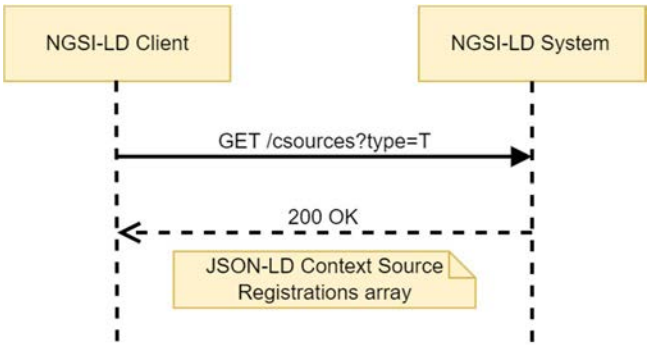

Figure 6.8.3.2-1: Query Context Source Registrations interaction

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 type names	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
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

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: csources/{registrationId}

6.9.1 Description

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

6.9.2 Resource definition

Resource URI:

- /csources/{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.

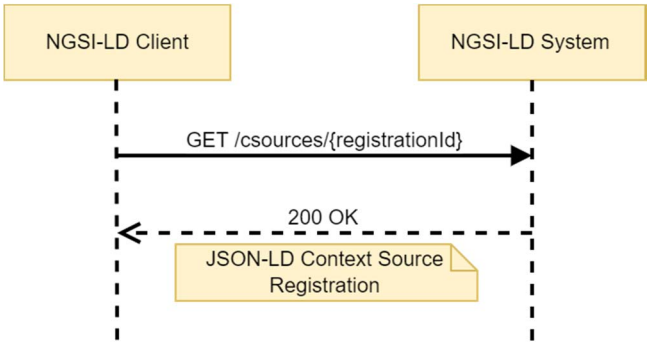

Figure 6.9.3.1-1: Retrieve Context Source Registration interaction

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 input payload. 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.

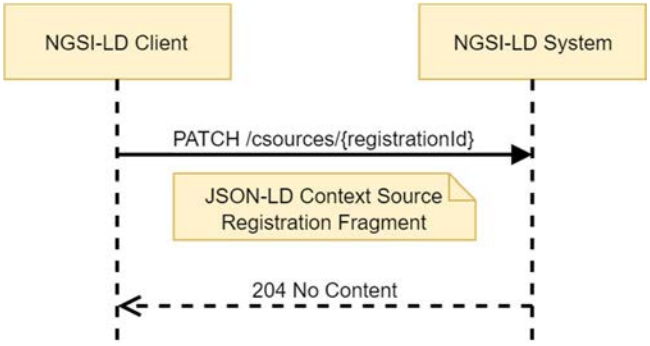

Figure 6.9.3.2-1: Update Context Source Registration interaction

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.

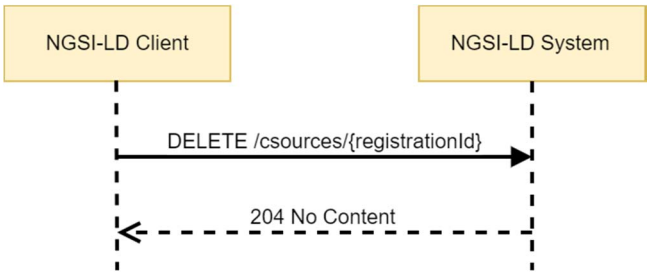

Figure 6.9.3.3-1: Delete Context Source Registration interaction

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 a 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 input payload. 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.

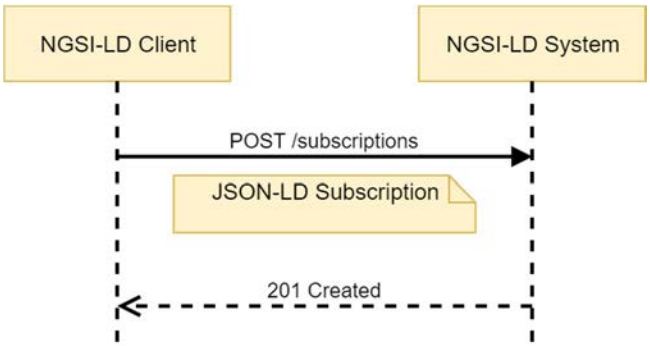

Figure 6.10.3.1-1: Create Subscription interaction

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 output payload. Figure 6.10.3.2-1 shows the Query Subscriptions interaction.

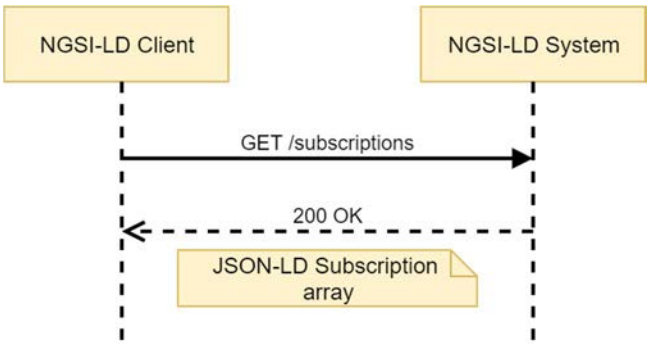

Figure 6.10.3.2-1: Query Subscriptions interaction

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 a 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.

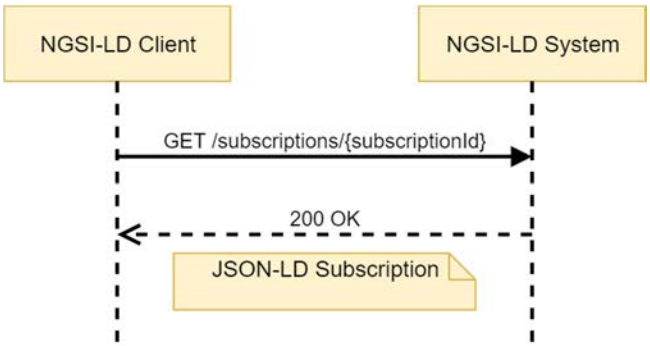

Figure 6.11.3.1-1: Retrieve Subscription interaction

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.

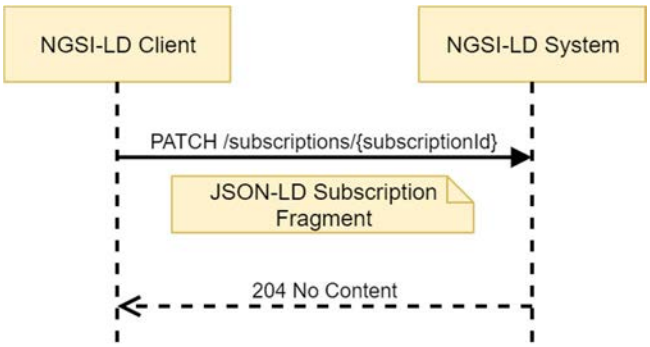

Figure 6.11.3.2-1: Update Subscription interaction

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.

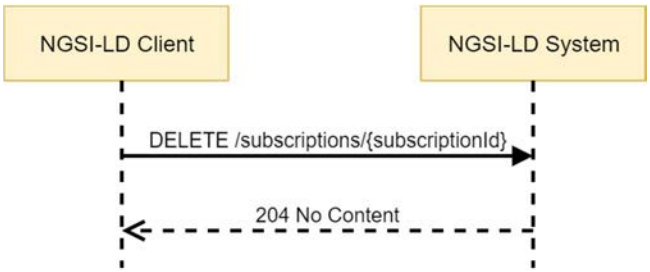

Figure 6.11.3.3-1: Delete Subscription interaction

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 a 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 5.11.8, taking the context source registration subscription to be created from the HTTP request input payload. 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.

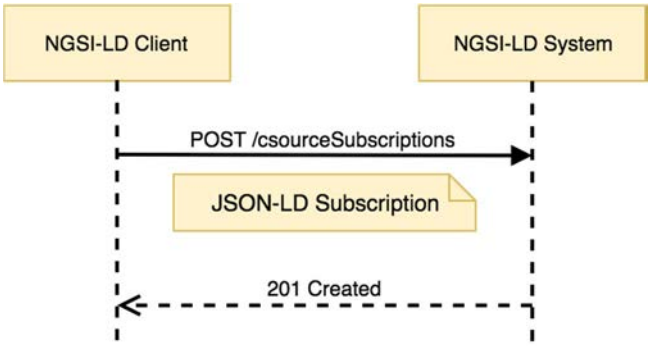

Figure 6.12.3.1-1: Create Context Source Registration Subscription interaction

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 output payload. Figure 6.12.3.2-1 shows the Query Context Source Registration Subscriptions interaction.

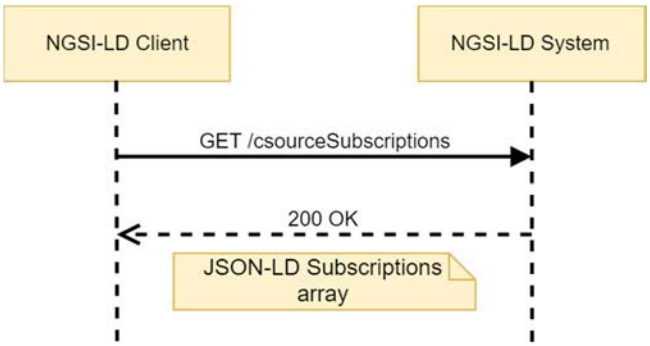

Figure 6.12.3.2-1: Query Context Source Registration Subscriptions interaction

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 a 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.

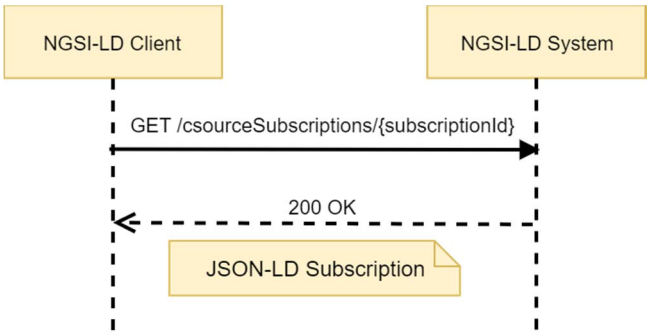

Figure 6.13.3.1-1: Retrieve Context Source Registration Subscription interaction

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.

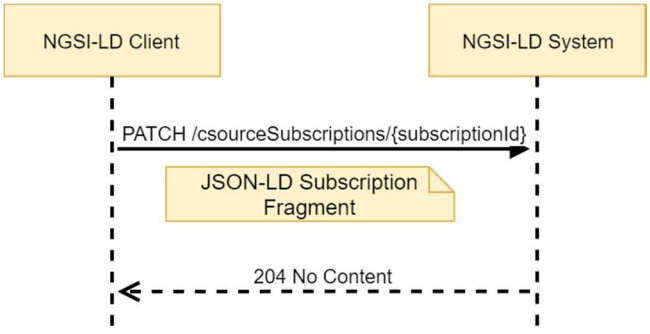

Figure 6.13.3.2-1: Update Context Source Registration Subscription interaction

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.

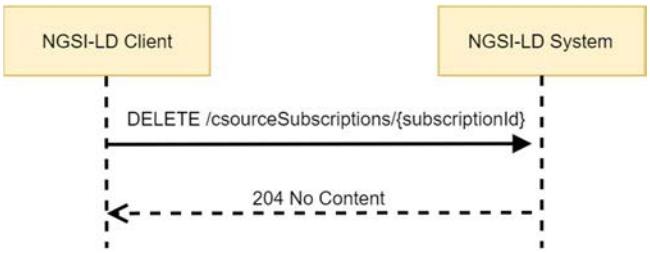

Figure 6.13.3.3-1: Delete Context Source Registration Subscription interaction

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.

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].

```
{
"id": "@id",
"type": "@type",
"value": "http://uri.etsi.org/ngsi-ld/hasValue",
"object": {
    "@id": "http://uri.etsi.org/ngsi-ld/hasObject",
    "@type":"@id"
},
"Property": "http://etsi.org/nsgi-ld/Property",
"Relationship": "http://uri.etsi.org/ngsi-ld/Relationship",
"DateTime": "http://uri.etsi.org/ngsi-ld/DateTime",
"Date": "http://uri.etsi.org/ngsi-ld/Date",
"Time": "http://uri.etsi.org/ngsi-ld/Time",
"createdAt": {
    "@id": "http://uri.etsi.org/ngsi-ld/createdAt",
    "@type": "DateTime"
},
"modifiedAt": {
    "@id": "http://uri.etsi.org/ngsi-ld/modifiedAt",
    "@type": "DateTime"
},
"observedAt": {
    "@id": "http://uri.etsi.org/ngsi-ld/observedAt",
    "@type": "DateTime"
},
"unitCode": "http://uri.etsi.org/ngsi-ld/unitCode",
"location": "http://uri.etsi.org/ngsi-ld/location",
"observationSpace": "http://uri.etsi.org/ngsi-ld/observationSpace",
"operationSpace": "http://uri.etsi.org/ngsi-ld/operationSpace",
"GeoProperty": "http://uri.etsi.org/ngsi-ld/GeoProperty",
"TemporalProperty": "http://uri.etsi.org/ngsi-ld/TemporalProperty",
"ContextSourceRegistration": "http://uri.etsi.org/ngsi-ld/ContextSourceRegistration",
"Subscription": "http://uri.etsi.org/ngsi-ld/Subscription",
"Notification": "http://uri.etsi.org/ngsi-ld/Notification",
"ContextSourceNotification": "http://uri.etsi.org/ngsi-ld/ContextSourceNotification",
"title": "http://uri.etsi.org/ngsi-ld/title",
"detail": "http://uri.etsi.org/ngsi-ld/detail",
"idPattern": "http://uri.etsi.org/ngsi-ld/idPattern",
"name": "http://uri.etsi.org/ngsi-ld/name",
"description": "http://uri.etsi.org/ngsi-ld/description",
"information": "http://uri.etsi.org/ngsi-ld/information",
"timestamp": "http://uri.etsi.org/ngsi-ld/timestamp",
"expires": {
    "@id": "http://uri.etsi.org/ngsi-ld/expires",
    "@type": "DateTime"
},
"endpoint": "http://uri.etsi.org/ngsi-ld/endpoint",
"entities": "http://uri.etsi.org/ngsi-ld/entities",
"properties": {
    "@id": "http://uri.etsi.org/ngsi-ld/properties",
    "@type": "@id"
},
"relationships": {
    "@id": "http://uri.etsi.org/ngsi-ld/relationships",
    "@type": "@id"
},
"start": {
    "@id": "http://uri.etsi.org/ngsi-ld/start",
    "@type": "DateTime"
},
"end": {
    "@id": "http://uri.etsi.org/ngsi-ld/end",
    "@type": "DateTime"
},
"watchedAttributes":{
    "@id": "http://uri.etsi.org/ngsi-ld/watchedAttributes",
    "@type": "@id"
},
```

```
"timeInterval": "http://uri.etsi.org/ngsi-ld/timeInterval",
"q": "http://uri.etsi.org/ngsi-ld/q",
"geoQ": "http://uri.etsi.org/ngsi-ld/geoQ",
"notification": "http://uri.etsi.org/ngsi-ld/notification",
"status": "http://uri.etsi.org/ngsi-ld/status",
"throttling": "http://uri.etsi.org/ngsi-ld/throttling",
"geometry": "http://uri.etsi.org/ngsi-ld/geometry",
"coordinates": "http://uri.etsi.org/ngsi-ld/coordinates",
"georel": "http://uri.etsi.org/ngsi-ld/georel",
"attributes": {
    "@id": "http://uri.etsi.org/ngsi-ld/attributes ",
    "@type": "@id"
},
"format": "http://uri.etsi.org/ngsi-ld/format",
"timesSent": "http://uri.etsi.org/ngsi-ld/timesSent",
"lastNotification":{
    "@id": "http://uri.etsi.org/ngsi-ld/lastNotification",
    "@type": "DateTime"
},
"lastFailure":{
    "@id": "http://uri.etsi.org/ngsi-ld/lastFailure ",
    "@type": "DateTime"
},
"lastSuccess":{
    "@id": "http://uri.etsi.org/ngsi-ld/lastSuccess",
    "@type": "DateTime"
},
"uri": "http://uri.etsi.org/ngsi-ld/uri",
"accept": "http://uri.etsi.org/ngsi-ld/accept",
"subscriptionId": {
    "@id": "http://uri.etsi.org/ngsi-ld/subscriptionId",
    "@type": "@id"
},
"notifiedAt":{
    "@id": "http://uri.etsi.org/ngsi-ld/notifiedAt",
    "@type": "DateTime"
},
"data": "http://uri.etsi.org/ngsi-ld/data",
"triggerReason": "http://uri.etsi.org/ngsi-ld/triggerReason"
}
```

NOTE: Implementers can take advantage of the "@vocab" JSON-LD keyword to provide a terser representation of the Core @context.

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.

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.

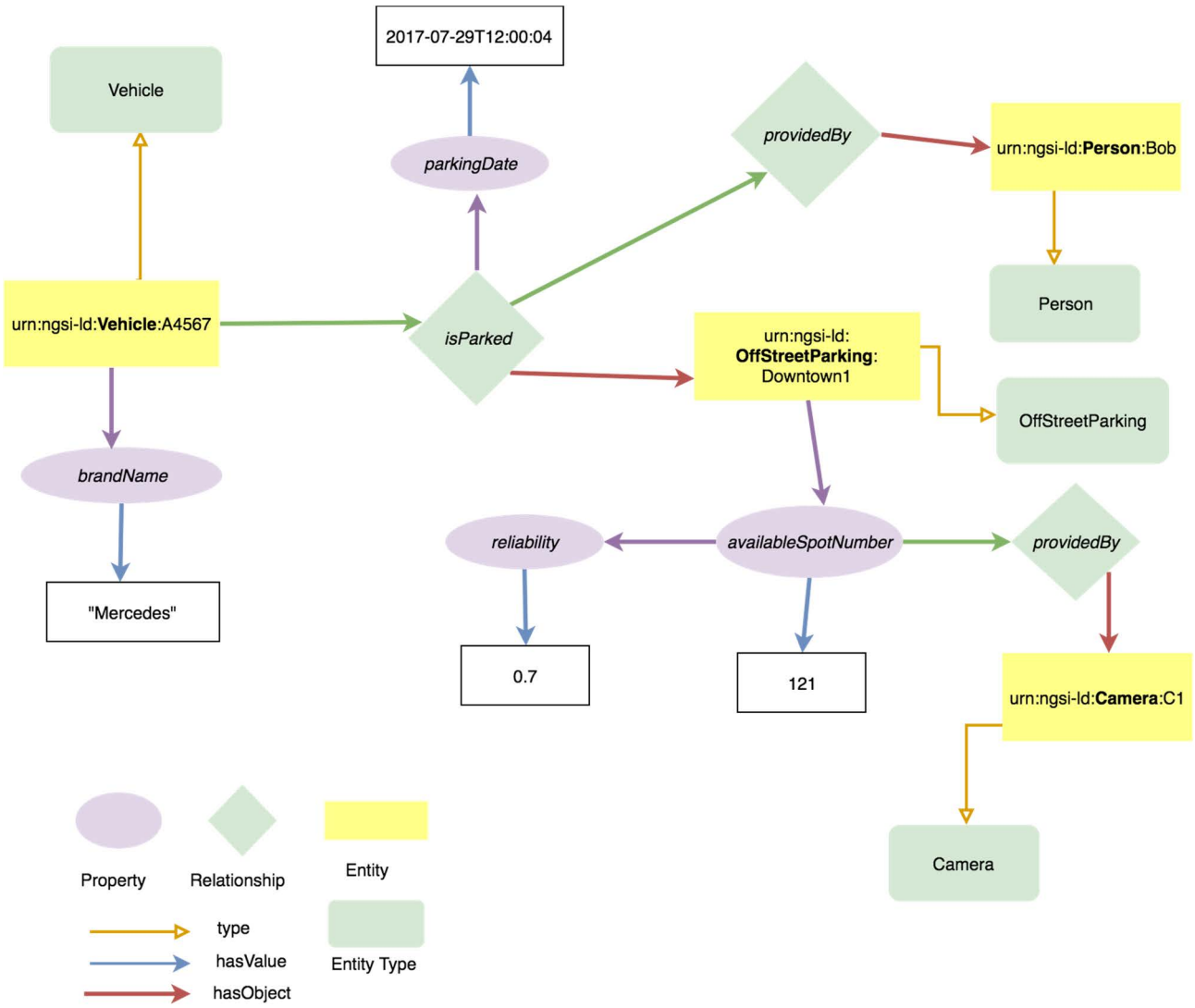

Figure C.2.1-1: Reference example

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": "2017-07-29T12:00:04",
        "providedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Person:Bob"
        }
    },
    "@context": [
        "http://uri.etsi.org/ngsi-ld/coreContext.jsonld",
        "http://example.org/cim/commonTerms.jsonld",
        "http://example.org/cim/vehicle.jsonld",
        "http://example.org/cim/parking.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://uri.etsi.org/ngsi-ld/coreContext.jsonld",
        "http://example.org/cim/commonTerms.jsonld",
        "http://example.org/cim/vehicle.jsonld",
        "http://example.org/cim/parking.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": "2017-07-29T12:05:02",
        "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://uri.etsi.org/ngsi-ld/coreContext.jsonld",
        "http://example.org/cim/parking.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://uri.etsi.org/ngsi-ld/coreContext.jsonld",
        "http://example.org/cim/parking.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": "http://uri.etsi.org/ngsi-ld/Property",
    "Relationship": "http://uri.etsi.org/ngsi-ld/Relationship",
    "value": "http://uri.etsi.org/ngsi-ld/hasValue",
    "object": {
        "@type": "@id",
        "@id": "http://uri.etsi.org/ngsi-ld/hasObject"
    },
    "observedAt": {
        "@type": "http://uri.etsi.org/ngsi-ld/DateTime",
        "@id": "http://uri.etsi.org/ngsi-ld/observedAt"
    },
    "location": "http://uri.etsi.org/ngsi-ld/location",
    "GeoProperty": "http://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": " 2017-11-29T14:53:15"
     },
    "@context": [
        "http://uri.etsi.org/ngsi-ld/coreContext.jsonld",
        "http://example.org/cim/commonTerms.jsonld",
        "http://example.org/cim/vehicle.jsonld",
        "http://example.org/cim/parking.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://uri.etsi.org/ngsi-ld/coreContext.jsonld",
        "http://example.org/cim/vehicle.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 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/entities/

Content-Type: application/ld+json

Content-Length: 556

<Insert Here JSON-LD Content from A.2.1>

C.5.2.2 HTTP Response

201 Created

Location: /ngsi-ld/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/entities/?type=Vehicle&q=brandName!=Mercedes&options=keyValues

Accept: application/ld+json

Link: <http://example.org/cim/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://uri.etsi.org/ngsi-ld/coreContext.jsonld",
            "http://example.org/cim/vehicle.jsonld"
        ]
    }
]
```

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 a 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 a 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 a 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

E.1 NGSI-LD Terms and categories: definitions

In addition to the definitions given in clause 3.1 of the present document, and in the context of this annex, further terms are defined as follows:

- **NGSI-LD Property Type:** An NGSI-LD Property Type is used to categorize an NGSI-LD Property as belonging to a class of similar properties. An NGSI-LD Property Type is identified by a URI.
- **NGSI-LD Relationship Type:** An NGSI-LD Relationship Type is used to categorize an NGSI-LD Relationship as belonging to a class of similar relationships. A NGSI-LD Relationship Type is identified by a URI.

E.2 Bridging Property graphs and RDF graphs

Superficially, RDF graphs could be seen as an alternative way to capture context information cast as a graph, but they start from a very different premise, placing the emphasis on semantic rather than structural information, mixing inter-individual with individual-to-class (instance to category) semantics. RDF is a supremely parsimonious meta-model, defining only, from first-order predicate logic, a basic (subject, predicate, object) triple as building block of a labelled graph, where a generic notion of property (instantiated in a predicate) does not distinguish relationships with property graph (though OWL does make a similar distinction object properties and data properties). RDF and the associated ontology languages (RDFS/OWL) do natively support all kinds of semantics but they cannot directly express more complex constructs that go beyond the expressivity of fist-order logic, such as properties of properties, properties of relationships and their derivatives, which a graph database natively supports. Property graphs are the implicit semi-formal data models underlying most graph databases, such as Neo4J, OrientDB, etc. They have gained widespread following as such, more in industry than in academia. They make it possible to attach properties to relationships, which RDF does not directly support, but they lack the standardization and formal underpinnings of RDF and do not interoperate directly with linked data and other RDF datasets. Also, they do not lend themselves to reasoning with RDF-based reasoning tools or querying with standard query languages such as SPARQL. Several solutions have been proposed to convert property graphs to RDF.

Reification is the default way to support this kind of derivative graphs, coming from property graphs, in RDF. Standard RDF reification [i.8] defines a new resource (with type rdf:statement) that encompasses a predicate jointly with its associated subject and object, i.e. a triple) This new statement resource is linked back to the original subject, object and predicate of the statement through a trio of properties with types rdf:subject, rdf:object and rdf:predicate, respectively. A total of 4 additional statements (corresponding to the infamous "reification quad") are thus required to fully define the reified statement as a resource, this only in order to make this resource the subject of other statement, making this a very cumbersome and heavyweight solution. Reification by way of blank nodes is a simpler way to achieve this:

Assuming that it is desired to reify *statement 1*, given as follows:

[StreetA → hasState → "30% busy"],

in order to make it the subject of another statement:

[statement 1] → reliability → "90%"

then it is only necessary to add a blank node:

StreetA → hasState → ***_blank_node***

_blank_node *→ reliability → "90%"*

_blank_node *→ hasValue → "30% busy"*

Moreover, the statement corresponding to the "reliability" property may further be reified in order to express more information concerning it.

This solution is especially convenient when the graph is serialized with JSON-LD because blank nodes do not explicitly appear in the textual serialized description, and actually show up only when it is represented as a graph. It is thus possible for a developer to generate the JSON-LD payload of an API in a form that is very similar to what he would have generated in plain JSON, or in the previous version of the OMA NGSI data model [i.3].

E.3 Tentative formal definition of NGSI-LD information model

E.3.1 Introduction

The NGSI-LD meta-model is defined as an RDF/RDFS/OWL ontology. Semantically, the meta-model is divided into two parts (accounting to two abstraction levels): **Core meta-model** and **Cross-domain meta-model**.

E.3.2 Core Meta-Model

The core meta-model provides the ontological definition of the primitive classes: Entity, Property, Relationship, and Value. These classes are the representative actors from the property graph model.

Entity, **Property**, and **Relationship** are classes, and are direct subclasses of the *rdfs:Resourse* class.

Value is a class whose instances are the only instances that can be associated to **Properties**. The class **Value** is disjoint with the **Entity**, the **Property**, and the **Relationship** classes. In principle, **Value** may be an *rdfs:Literal*, but may be also another class of nodes that is designed in the cross-domain meta-model (like **Geomotry** which is a GeoJSON expression as defined in this API document), or other classes that may be defined in a domain-specific meta-model by some NGSI-LD user. Thus, the only intrinsic restrictions on **Value** is that it is the *rdfs:range* of all properties, and it can be any class that is designed to be a value given that it is not an **Entity**, a **Property**, nor a **Relationship**. Further restrictions are defined in our meta-model on individual properties, like restricting the *rdfs:range* of the **location** property to be a **Geometry** value.

hasValue and **hasObject** are primitive RDF properties required by our adopted reification technique and supplementary properties. The core meta-model diagram defines the *rdfs:domain* (orange arrow) and the *rdfs:range* (green arrow) of each.

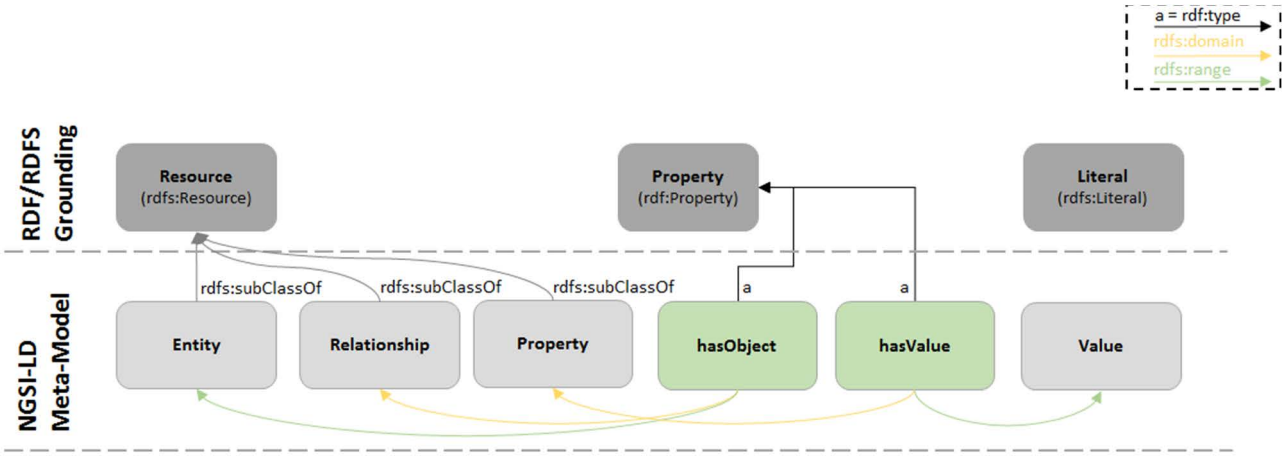

Figure E.3.2-1: NGSI-LD core meta-model diagram

E.3.3 Cross-Domain Meta-Model

The cross-domain meta-model provides definition of property types, relationship types, and value types that are considered to be general yet common tools for any system that is using NGSI-LD.

A set of rules for property types is defined as illustrated below. (*Example: observationSpace*).

For each property type **p**:

- the following rules hold:
  - **p** rdfs:subClassOf **Property**
    (Either direct or by inference, i.e. p may also be defined as an rdfs:subClassOf some class which is a subclass of Property.)
  - **p** rdf:type **p**
    (p is both a class which is the class of property blank node instances, and an instance of itself because it is used as a property that links subjects to the property blank nodes)
  - **p** owl:propertyChainAxiom (**p hasValue**)
    (Since reification uses a supplementary property hasValue with Properties, a shortcut is allowed for extracting data by only using the property p. This assertion means that any property p usage followed by a hasValue usage is semantically equivalent to using p alone. This allows implicitly inferring the values of properties without using the reification through blank nodes, but directly through the name of the property when no other information is needed.)
- the following rules may hold: (where restrictions are needed):
  - **p** rdfs:domain **C** *(Where **C** is a subclass of **Entity**, **Property**, or **Relationship**)*
    (This limits the usage of p on a certain class of Entities, Properties, or Relationships)
  - **p** rdfs:range **V** *(Where **V** is a subclass of **Value**)*
    (This limits the values that can be associated with p)

Note that some properties in NGSI-LD never follow the reification technique. These properties are direct instances (using *rdf:type*) of the class **Property** and they include: *observedAt*, *createdAt*, *modifiedAt*, *start*, *end* and *unitCode*.

Similar to property types, it is necessary to define the rules for relationship types.

For each relationship type **r**:

- the following rules hold:
  - **r** rdfs:subClassOf **Relationship**
    (Either direct or by inference, i.e. r may also be defined as an rdfs:subClassOf some class which is a subclass of Relationship.)
  - **r** rdf:type **r**
    (r is both a class which is the class of relationship blank node instances, and an instance of itself because it is used as a property that links subjects to the relationship blank nodes)
  - **r** owl:propertyChainAxiom (**r hasObject**)
    (Since reification uses a supplementary property hasObject with Relationships, it is allowed to define a shortcut for extracting data by only using the relationship r. This assertion means that any relationship r usage followed by a hasObject usage is semantically equivalent to using r alone. This allows implicitly inferring the values of relationships without using the reification through blank nodes, but directly through the name of the relationship when no other information is needed.)
- the following rules may hold: (where restrictions are needed):
  - **r** rdfs:domain **C1** *(Where **C1** is a subclass of **Entity**, **Property**, or **Relationship**)*
    (This limits the usage of r on a certain class of Entities, Properties, or Relationships)
  - **r** rdfs:range **C2** *(Where **C2** is a subclass of **Entity**)*
    (This limits the Entities that can be associated with r)

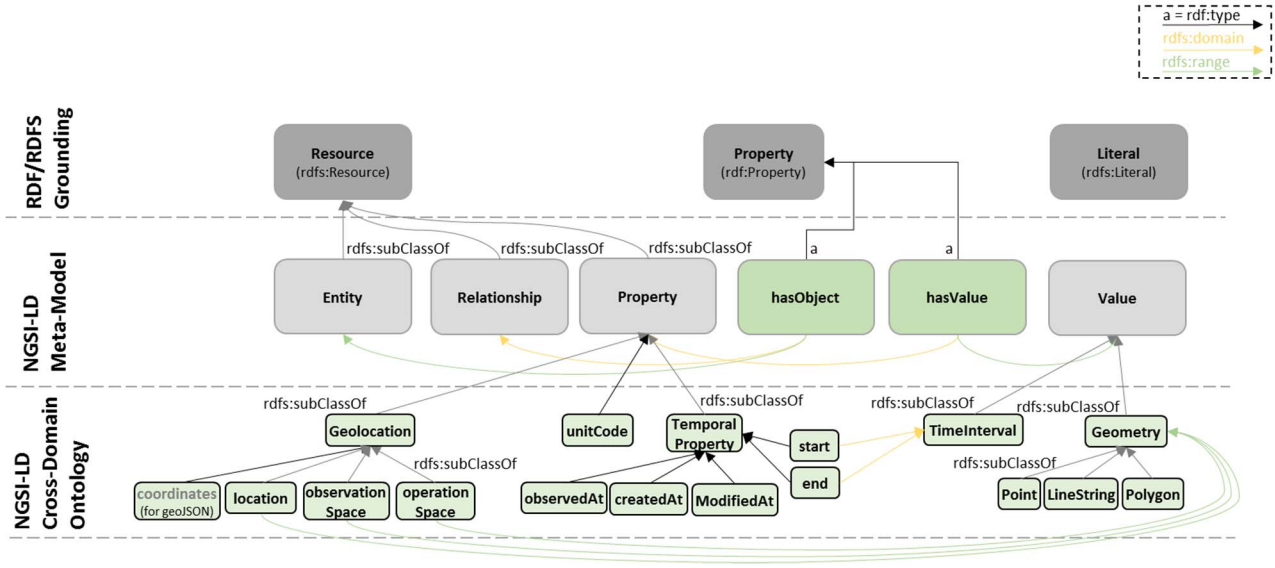

Figure E.3.3-1: NGSI-LD cross-domain meta-model diagram

E.4 Example

Figure E.4-1 shows a simple example, with the corresponding categories defined, reified according to the method suggested.

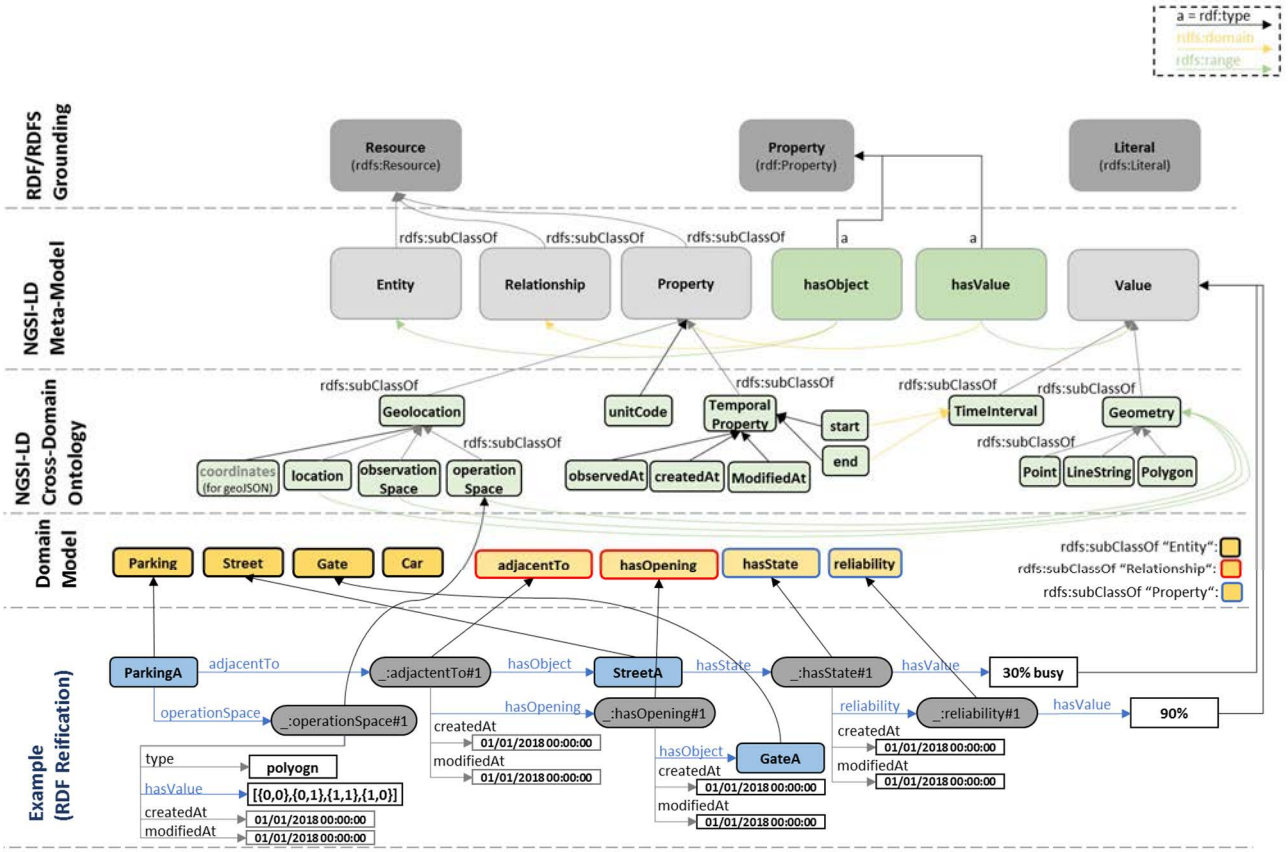

Figure E.4-1: NGSI-LD Example

E.5 Complete Ontology in Turtle RDF Syntax

```
@prefix : <http://cim.example.org/cim-meta-model#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://cim.example.org/cim-meta-model> .

<http://cim.example.org/cim-meta-model> rdf:type owl:Ontology .

#################################################################
#    Object Properties
#################################################################

###  http://cim.example.org/cim-meta-model#coordinates
:coordinates rdf:type owl:ObjectProperty .


###  http://cim.example.org/cim-meta-model#hasObject
:hasObject rdf:type owl:ObjectProperty ;
           rdfs:domain :Relationship ;
           rdfs:range :Entity .


###  http://cim.example.org/cim-meta-model#hasValue
:hasValue rdf:type owl:ObjectProperty ;
          rdfs:domain :Property ;
          rdfs:range :Value .


###  http://cim.example.org/cim-meta-model#location
:location rdf:type owl:ObjectProperty ;
          rdfs:range :Geometry ;
          owl:propertyChainAxiom ( :location
                                   :hasValue
                                 ) .


###  http://cim.example.org/cim-meta-model#observationSpace
:observationSpace rdf:type owl:ObjectProperty ;
                  rdfs:range :Geometry ;
                  owl:propertyChainAxiom ( :observationSpace
                                           :hasValue
                                         ) .


###  http://cim.example.org/cim-meta-model#operationSpace
:operationSpace rdf:type owl:ObjectProperty ;
                rdfs:range :Geometry ;
                owl:propertyChainAxiom ( :operationSpace
                                         :hasValue
                                       ) .


###  http://www.w3.org/2000/01/rdf-schema#domain
rdfs:domain rdf:type owl:ObjectProperty .


###  http://www.w3.org/2000/01/rdf-schema#range
rdfs:range rdf:type owl:ObjectProperty .


#################################################################
#    Data properties
#################################################################

###  http://cim.example.org/cim-meta-model#createdAt
:createdAt rdf:type owl:DatatypeProperty ;
           rdfs:range xsd:dateTime .


###  http://cim.example.org/cim-meta-model#end
:end rdf:type owl:DatatypeProperty ;
     rdfs:domain :TimeInterval ;
```

```
     rdfs:range xsd:dateTime .


###  http://cim.example.org/cim-meta-model#modifiedAt
:modifiedAt rdf:type owl:DatatypeProperty ;
            rdfs:range xsd:dateTime .


###  http://cim.example.org/cim-meta-model#observedAt
:observedAt rdf:type owl:DatatypeProperty ;
            rdfs:range xsd:dateTime .


###  http://cim.example.org/cim-meta-model#start
:start rdf:type owl:DatatypeProperty ;
       rdfs:domain :TimeInterval ;
       rdfs:range xsd:dateTime .


###  http://cim.example.org/cim-meta-model#unitCode
:unitCode rdf:type owl:DatatypeProperty ;
          rdfs:range xsd:string .


#################################################################
#    Classes
#################################################################

###  http://cim.example.org/cim-meta-model#Entity
:Entity rdf:type owl:Class ;
        rdfs:subClassOf rdfs:Resource ;
        owl:disjointWith :Value .


###  http://cim.example.org/cim-meta-model#Geolocation
:Geolocation rdf:type owl:Class ;
             rdfs:subClassOf :Property .


###  http://cim.example.org/cim-meta-model#Geometry
:Geometry rdf:type owl:Class ;
          rdfs:subClassOf :Value .


###  http://cim.example.org/cim-meta-model#LineString
:LineString rdf:type owl:Class ;
            rdfs:subClassOf :Geometry .


###  http://cim.example.org/cim-meta-model#Point
:Point rdf:type owl:Class ;
       rdfs:subClassOf :Geometry .


###  http://cim.example.org/cim-meta-model#Polygon
:Polygon rdf:type owl:Class ;
         rdfs:subClassOf :Geometry .


###  http://cim.example.org/cim-meta-model#Property
:Property rdf:type owl:Class ;
          rdfs:subClassOf rdfs:Resource ;
          owl:disjointWith :Value .


###  http://cim.example.org/cim-meta-model#Relationship
:Relationship rdf:type owl:Class ;
              rdfs:subClassOf rdfs:Resource ;
              owl:disjointWith :Value .


###  http://cim.example.org/cim-meta-model#TemporalProperties
:TemporalProperties rdf:type owl:Class ;
                    rdfs:subClassOf :Property .


###  http://cim.example.org/cim-meta-model#TimeInterval
:TimeInterval rdf:type owl:Class ;
```

```
              rdfs:subClassOf :Value .


###  http://cim.example.org/cim-meta-model#Value
:Value rdf:type owl:Class .


###  http://cim.example.org/cim-meta-model#end
:end rdf:type owl:Class ;
     rdfs:subClassOf :TemporalProperties .


###  http://cim.example.org/cim-meta-model#location
:location rdf:type owl:Class ;
          rdfs:subClassOf :Geolocation .


###  http://cim.example.org/cim-meta-model#observationSpace
:observationSpace rdf:type owl:Class ;
                  rdfs:subClassOf :Geolocation .


###  http://cim.example.org/cim-meta-model#operationSpace
:operationSpace rdf:type owl:Class ;
                rdfs:subClassOf :Geolocation .


###  http://cim.example.org/cim-meta-model#start
:start rdf:type owl:Class ;
       rdfs:subClassOf :TemporalProperties .


###  http://www.w3.org/2000/01/rdf-schema#Resource
rdfs:Resource rdf:type owl:Class .


#################################################################
#    Individuals
#################################################################

###  http://cim.example.org/cim-meta-model#coordinates
:coordinates rdf:type owl:NamedIndividual ,
                      :Geolocation .


###  http://cim.example.org/cim-meta-model#createdAt
:createdAt rdf:type owl:NamedIndividual ,
                    :TemporalProperties .


###  http://cim.example.org/cim-meta-model#end
:end rdf:type owl:NamedIndividual ,
              :TemporalProperties .


###  http://cim.example.org/cim-meta-model#location
:location rdf:type owl:NamedIndividual ,
                   :Geolocation .


###  http://cim.example.org/cim-meta-model#modifiedAt
:modifiedAt rdf:type owl:NamedIndividual ,
                     :TemporalProperties .


###  http://cim.example.org/cim-meta-model#observationSpace
:observationSpace rdf:type owl:NamedIndividual ,
                           :Geolocation .


###  http://cim.example.org/cim-meta-model#observedAt
:observedAt rdf:type owl:NamedIndividual ,
                     :TemporalProperties .


###  http://cim.example.org/cim-meta-model#operationSpace
:operationSpace rdf:type owl:NamedIndividual ,
                         :Geolocation .
```

```
###  http://cim.example.org/cim-meta-model#start
:start rdf:type owl:NamedIndividual ,
               :TemporalProperties .


###  http://cim.example.org/cim-meta-model#unitCode
:unitCode rdf:type owl:NamedIndividual ,
                   :Property .


###  Generated by the OWL API (version 4.2.8.20170104-2310) https://github.com/owlcs/owlapi
```

Annex F (informative): Gap analysis on the relationship of NGSI-LD and general triple-based queries

To fully make use of the expressiveness of the NGSI-LD information model, the NGSI-LD API may continue to evolve after the first release and support more query patterns by taking the general triple query pattern as a reference. In fact, JSON-LD is a serialization format for linked data essentially based on the RDF meta model.

RDF data is organized as triples. A unit of RDF data is a triple, which is a statement with three constituent parts: a subject, a predicate, and an object, such as sky (subject) is (predicate) blue (object). A statement is denoted as stat, and represented as stat = <s, p, o>, where s, p, o are respectively the subject, predicate and object of the statement *stat*. The triple patterns that the NGSI-LD system could benefit from consists of 1-wise pattern, 2-wise pattern and 3-wise pattern, that respectively query 1, 2 or 3 elements of a statement. The triple query patterns are presented in table F-1.

Table F-1: Triple Query Patterns

Query Patterns	Triple: <s, p, o>		
1-wise	<**s**, p, o>	<s, **p**, o>	<s, p, **o**>
2-wise	<**s**, **p**, o>	<**s**, p, **o**>	<s, **p**, **o**>
3-wise	<**s**, **p**, **o**>		

In table F-1, seven query patterns are identified as basic triple query patterns, and the red letters represent the elements that a query pattern addresses. For example, in the 1-wise pattern where a Context Consumer only indicates any subject *s*, the Context Consumer can query all information related to a subject. For instance, if the subject is a person, Tom, by such query all triples related to the person Tom will be returned. Particularly, in the 3-wise pattern where the Context Consumer indicates the subject, predicate and object in one query, the Context Consumer can query if a specific statement exists in the data base, and the returned result is the same triple if the specific statement exists, otherwise no result is returned. The query patterns for RDF data, i.e. triple pattern queries, are able to answer any type of questions, so that platforms can easily support various query requirements and federate different existing platforms.

Table F-2 identifies the mapping from the general concepts of subject, predicate and object, to concepts in the NGSI-LD information model. For example, the predicate in NGSI-LD can be either a Relationship or a Property. Particularly, data based on JSON-LD serialization hides the representation of blank nodes in triples; in a pure RDF-based implementation with triple-stores, blank node can be either subject or object in table F-2.

Table F-2: NGSI-LD mapping with subject, predicate and object

Subject (s)	Predicate (p)	Object (o)
- Entity	- Relationship - Property	- Entity - Value

Table F-3 presented the gap analysis summary between current NGSI-LD APIs and the general triple query patterns. For each triple query pattern, the Context Consumer identifies one query example with specific value, and its mapping to current NGSI-LD support.

From table F-3 it can be deduced 3 of 7 triple patterns are supported by the current NGSI-LD API. It is noteworthy that the gap analysis is served as a first step for future improvement, which includes basic triple query patterns but does not include all complex query patterns on graph model such as query on a sub graph structure, graph construction, etc.

The identified gaps in the present annex help to understand the potential enhancement to be made in the following releases NGSI-LD APIs, in order to provide users with more flexible query interfaces. In later work it will be further investigated the gaps identified and decide the partial or full patterns that to be supported by NGSI-LD APIs. Generally, the choice will be made by balancing costs and benefits e.g. query flexibility, efficiency, usability and impact. Flexible query patterns may influence the API performance depending on the real use cases; especially data characteristics and architectural organization will be taken into account. In a dataset stored in a centralized architecture where entity relationships and values barely change (e.g. birth information in a region), all patterns could be supported; on the other hand, for one or more datasets organized in a distributed or federated architecture, and entity relationships or values that frequently change, query patterns on entity values and relationships cannot be efficiently supported.

Table F-3: Gap summary between current APIs and general triple query patterns

< s, p, o >	**< s, p, o >**	**< s, p, o >**
- Entity Id: person32, to query all information about this *person32*	- Relationship: *connectedTo*, to query all entities that are connected with each other, e.g. in blockchain	- value: sick, to query all entities with properties that have the value "sick"
/entities/{entityId}	Not supported	Not supported
< s, p, o >	**< s, p, o >**	**< s, p, o >**
- Entity Id: person32 - Relationship: *hasContact* to query all objects corresponding to *person32* and *hasContact* Relationship	- Entity Id: person32 - Entity Id: person33 to query all Relationships between *person32* and *person33*	- Relationship: *applyFor* - Entity Id: *CIMCertification* to query all entities that apply for CIM Certification
/entities/{entityId}/attrs/{attrId}	Not supported	Not supported
< s, p, o >		
- Entity Id: *person32*, - Relationship: *isEmployerOf*, - Entity Id: *company20* to query <person32, isEmployerOf, company20>, if the person32 is an employer of the company20.		
/entities/{entityId}/attrs/{attrId}.value		

Annex G (informative): Roadmap of Functionalities

Table G-1 describes the roadmap of functionalities to be considered for the **final** NGSI-LD API specification (Release 1). They are divided into three priority levels:

- *High priority*. Functionalities that the ISG CIM has committed to address (if after further study it is concluded that they are feasible, both implementation-wise and specification-wise).
- *Medium priority*. Functionalities that is highly recommended to deal with and, provided enough resources are available, they are expected to be addressed.
- *Low priority*. Functionalities that are not a priority for ISG CIM but which can be considered in the discussions if good technical contributions come across.

Table G-1: Summary of future functionalities to be addressed

Feature Name/Gap	Description	Priority
Complete expression language for queries	Additional logical operators: *not*, *or* Arbitrary levels of bracketing: (,) Basic mathematical operations (applied to Attributes/ aggregate functions): +, -, *, / Null tests: *is null*, *not null*	High
Pagination of results	Enable pagination of query results involving NGSI-LD elements (Entities, Context Source Registrations, Context Subscriptions). Study the definition of cursors to improve iteration processes. Limit maximum number of entities that are returned in a list	High
Complete Historical data functionality	Apart from a generic interface between Context Producers and Consumers for 'real-time' or 'near-real-time' usage many applications would benefit from the ability to query historical snapshots of entity data e.g. in identifying a seasonal pattern to data, including: • Allow representing time series that show temporal evolution • Define a simplified representation for time series data to facilitate its consumption by applications, study what oneM2M has already defined • Applications should be able to retrieve instances of entities over a defined period of time • There should also be support for retrieving at intervals e.g. Entities at one or more specific times of day e.g. 6pm, hourly, daily (different granularities) • Retrieve Entity Property Value stored at/before given date/time. Result data includes actual date/time stored at server (except where aggregate operations used) • Query Entity (sensor/ context data points) recorded over a specified date/ time range • Query Entity (sensor/ context data points) recorded at defined intervals over a specified date/ time range • Result ordering rules based on Entity creation date/time ascending/ descending. Multiple ordering criteria can be specified	High
Query Subscriptions	Enable queries over Subscriptions so that they can be filtered out properly	High
Extend Subscription functionality	Allow specifying the maximum limit of Entities provided when notifying Provide the capability of subscribing to higher level events (Entity creation, Entity modification, Entity deletion, etc.) Use HTTP PUT method for delivering notifications	High
Extend Context Source Registration Functionality	Allow adding custom, query-able members to Context Source Registrations. For instance, extra metadata about the Context Source (accuracy of the information provided, associated place, etc.)	High
Entity versioning mechanism	Specify clearly mechanisms to support the evolution of information models	High
Operations to manage historical time series	Add instances to time series Modify instances of time series Remove instances from time series	High

Feature Name/Gap	Description	Priority
Security	Add mechanisms to the API to enable all the Security requirements and findings required by the corresponding work item	High
Multiple Typing of entities, relationships and properties	Future versions of the data model should support the possibility for an instance of entities, relationships and properties to inherit multiple categories/classes as defined by the cross-domain ontology and, potentially, domain-specific ontologies	High
Queries over deeply structured data	Complete the query language to allow for Entity selection based on sub-elements of an Entity/Attribute Develop a more sophisticated specification for returning only those Attributes/ sub-elements required by the consuming application	Medium
Queries based on triple patterns as suggested by annex F	Study how triple patterns could be supported as an extended and more versatile query language. What are the costs? Are they really feasible?	Medium
Aggregate operators for queries	In many applications it is useful to be able to use aggregate functions provided by the database layer rather than the application having to retrieve all records and then perform the aggregate function. This is also normally more resource efficient both on the server and application side. The query specification would be improved by the support of the following: • Return a count of matching records (multiple Entities/ historical Entities) • Return a sum of an Attribute's value across matching records (multiple Entities/ historical Entities) Return the average, minimum or maximum value of an attribute's value across matching records (multiple Entities/ historical Entities)	Medium
Order by	Order matched Entities by Attribute Name with ordering directives (ascending, descending). Multiple ordering criteria can be specified. Study implications in distributed architectures	Medium
Batch operations	Specify operations that allow to manipulate multiple entities at the same time. Creation, Deletion, Modification, etc.	Medium
Privacy	Add mechanisms to the API to enable all the Privacy requirements and findings required by the corresponding work item	Medium
Internationalization	Support internationalized content and queries exploiting JSON-LD capabilities for string internationalization	Medium
Extended Geo-queries	Further study of other operators and geo-relationships for a richer set of geo-queries	Medium
Queries traversing the graph through Relationships	API allows retrieval of explicitly Linked Entities API allows specification of which Linked Entities are to be returned API allows specification of which Linked Entity fields are to be returned Entity references returned which allow direct retrieval Where there are Relationships between historical Entities there needs to be support to retrieve the related Entity data at the same historical time point	Low
Other output formats	Numerical analytics and machine learning generally require data to be provided in a uniform, tabular format such as available through Excel worksheets, Comma Separated Value (CSV) text files or Tab Separated Value (TSV) text files. Whilst the JSON format results is extremely useful for processing within applications it carries an overhead which means the data cannot be used immediately within numerical analytics or machine learning applications without further processing. Adding support for alternate output formats for results would be an advantage	Low

Annex H (informative): Open Issues

Table H-1 describes the open issues related to the present document. The ETSI ISG CIM is expected to fix them in the final NGSI-LD API Specification. It is noteworthy that some of them may overlap with the roadmap presented in annex G, particularly those related to "historical data functionality".

Table H-1: Summary of open issues

Issue Number	Description
ISSUE-001	Should the NGSI-LD query language rely on regular expressions or just provide an SQL-like pattern language?
ISSUE-002	Should NGSI-LD generalize Subscriptions so that any NGSI-LD element (Entity, Context Source Registration, Subscription) can be subscribed to? Is it possible to get rid of the duplication of Context Source Subscription?
ISSUE-003	Clarify how the update operations for Entities work. Should usage of JSON Merge Patch, which does not allow to have *null* values in Entities, continue? Should the JSON Patch (IETF RFC 6902 [i.10]) be used instead?
ISSUE-004	*TemporalProperty* on arbitrary properties, is it really needed?
ISSUE-005	Clarify how historical data collection works, making it clear that temporal queries are queries associated to Entity's history In the absence of temporal dimension regular queries operate over the last value / object of Attributes (Properties or Relationships) How temporal queries combine with filter queries? Proposal to be discussed: If a temporal query is used then filter query operates over the whole history under scope as per the temporal conditions. Think more about use cases combining filter queries and historical queries.
ISSUE-006	With regards to temporal queries: what happens if data is recorded every second but the query should obtain only data every hour over the last 5 days. If the query asks for data from the last 5 days then too much data will be provided to the application, which then ignores the vast majority.
ISSUE-007	In the Subscription data type the *status* member is overloaded, i.e. it is used both for system-provided information and for changing the subscription status. It is needed to reconsider it and probably define two different members (one for system provided details) and the other for controlling the activation status of a subscription.
ISSUE-008	In the abstract description of API operations use tables for describing input and output parameters instead of bullet points. Consider defining a logical name for each parameter.
ISSUE-009	In the *CsourceRegistration* data type consider renaming the member *timestamp* to something better, renaming it to "observedTimeRange" in *CsourceRegistration* data type.
ISSUE-010	In data types that make use of array collections ([]) clarify in the definition what happens if the collection is empty or includes *null* values (or if it is even allowed).
ISSUE-011	When returning a collection of Entities as application/json+ld, instead of encoding the @context in each entity, provide a single @context that applies to any Entity. This can be a bit more difficult to be generated when the result set of Entities pertain to different vocabularies.
ISSUE-012	Should the result of a Context Source Registration Query filter out those constituent parts of registration records that do not actually match?. Related to this issue: Should it be recommended to simplify the *information* member of the *CsourceRegistration* structure so that the limit is set to one *EntityInfo* and to one set of Properties and Relationships?
ISSUE-013	Refactor Context Source Registration matching so that it is centralized in one section of the document, avoiding duplication. Check the document in general to see if it is a better strategy to have always backwards references.
ISSUE-014	It is needed a mechanism, error code or similar, intended to flag the fact that a query cannot be resolved or that would take too much time.
ISSUE-015	Align Context Source Registrations clauses with the rest of operation descriptions.

Issue Number	Description
ISSUE-016	Clarify what happens when an attribute is updated, are all implementations required to handle time series of values? What happens if an implementation cannot do it?
ISSUE-017	Use of IRIs in the specification.
ISSUE-018	Physical things vs digital representations issue. Clarify it in the document.

Annex I (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: 2018-02-09T11:00:00.

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 J (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:
José Manuel Cantera Fonseca, FIWARE Foundation e.V.

Other contributors:
Patrick Guillemin, ETSI.

Martin Bauer, Lindsay Frost, NEC, whose work was also supported by funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No [723156] (WISE-IoT) and No [732240] (SynchroniCity).

Gilles Privat, Abdullah Abbas, Orange.

Wenbin Li, EGM, whose work was also supported by funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No [723156] (WISE-IoT).

David Fernández, HOP Ubiquitous, whose work was also supported by funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No [723174] (SmartSDK) and No [732240] (SynchroniCity).

Mike Fisher, BT.

Alistair Wright, GSMA.

SeungMyeong Jeong, KETI.

Annex K (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

History

Document history		
V1.1.1	April 2018	Publication