ETSI GS CIM 009 V1.7.1 (2023-06)

Context Information Management (CIM); NGSI-LD API

Disclaimer

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

Reference
RGS/CIM-009v171

Keywords
API, architecture, digital twins, GAP, information model, interoperability, NGSI-LD, smart agriculture, smart city, smart water, WoT

ETSI

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

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

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

Contents

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

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

Foreword

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

Modal verbs terminology

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

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

Executive summary

The present document formally describes the Context Information Management API (NGSI-LD) Specification. The Context Information Management API allows users to provide, consume and subscribe to context information in multiple scenarios and involving multiple stakeholders. Context information is modelled as attributes (properties and relationships) of context entities, also referred to as "digital twins", representing real-world assets. 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 NGSI-LD API Specification. This Context Information Management API allows users to provide, consume and subscribe to context information in multiple scenarios and involving multiple stakeholders. Context information is modelled as attributes of context entities, also referred to as "digital twins", representing real-world assets (e.g. a bus in a city or a luggage claim ticket). Because of that, the NGSI-LD API is often used to bring standardized access to digital twin data.

The ongoing status of the NGSI-LD API can be found in [i.17].

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

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

1 Scope

The purpose of the present document is the definition of a standard API for Context Information Management (NGSI-LD API) enabling close to real-time (right-time) access to context/digital twin information coming from many different sources (not only IoT data sources). The present document defines how such an API enables applications to perform updates on context, register context providers which can be queried to get updates on context, query information on current and historic context information and subscribe to receive notifications of context changes. The criteria for choice of the API characteristics are based on requirements resulting from the Use Cases ETSI GR CIM 002 [i.1] and other work items ETSI GR CIM 007 [i.2] and ETSI GS CIM 006 [i.8] on security and on the information model. The present document supersedes prior versions, including ETSI GS CIM 004 [i.16].

2 References

2.1 Normative references

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

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

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

The following referenced documents are necessary for the application of the present document.

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

[2] W3C® Recommendation 16 July 2020: "JSON-LD 1.1 - A JSON-based Serialization for Linked Data".

[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 8288: "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 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".

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

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

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

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

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

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

[20] IANA Registry of Link Relation Types.

[21] ECMA 262 Specification: "ECMAScript® 2022 language specification".

[22] The Unicode Consortium: "The Unicode Standard".

[23] IETF RFC 3987: "Internationalized Resource Identifiers (IRIs)".

[24] OASIS Standard: "MQTT Version 3.1.1 Plus Errata 01". Edited by Andrew Banks and Rahul Gupta. 10 December 2015.

[25] OASIS Standard: "MQTT Version 5.0". Edited by Andrew Banks, Ed Briggs, Ken Borgendale and Rahul Gupta. 07 March 2019.

[26] IETF RFC 7240: "Prefer Header for HTTP".

[27] IETF RFC 7230: "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing".

[28] IETF RFC 5646: "Tags for Identifying Languages".

[29] IETF RFC 3282: "Content Language Headers".

[30] IETF RFC 7234: "Hypertext Transfer Protocol (HTTP/1.1): Caching".

[31] IETF RFC 7233: "Hypertext Transfer Protocol (HTTP/1.1): Range Requests".

[32] IANA: "Hypertext Transfer Protocol (HTTP) Warn Codes".

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 (V1.1.1): "Context Information Management (CIM); Use Cases (UC)".

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

[i.3] OMA-TS-NGSI_Context_Management-V1_0-20120529-A: "NGSI Context Management".

[i.4] ETSI TS 103 264 (V3.1.1): "SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping".

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

[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.

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

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

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

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

[i.12] OpenAPI™ Specification.

[i.13] NGSI-LD JSON Schemas.

[i.14] NGSI-LD OpenAPI™ Specification.

[i.15] NGSI-LD Examples.

[i.16] ETSI GS CIM 004 (V1.1.2): "Context Information Management (CIM); Application Programming Interface (API)".

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

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

[i.19] MQTT URI Scheme.

[i.20] GeoJSON-LD 1.0: Base context for processing GeoJSON according to the JSON-LD processing model.

[i.21] ETSI GR CIM 008: "Context Information Management (CIM); NGSI-LD Primer".

[i.22] IoT Agent Library.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

NOTE 1: 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.

NOTE 2: The use of URI in the context of the present document also includes the use of International Resource Identifiers (IRIs) as defined in IETF RFC 3987 [23], which extends the use of characters to Unicode characters [22] beyond the ASCII character set, enabling the support of languages other than English.

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

NGSI-LD Attribute Instance (in case of temporal representation of NGSI-LD Entities): reference to an NGSI-LD Attribute, at a specific moment in time of its temporal evolution, usually identified by its instanceId

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 Context Source Registrations: description of the information that can be provided by a Context Source, which is used when registering the Context Source with the Context Registry

NGSI-LD Core API: core part of the NGSI-LD API that has to be implemented by all Brokers, including operations for providing or managing Entities and Attributes, operations for consuming Entities and checking which Entity Types and Attributes Entities are available in the system and operations for subscribing to Entities, receiving notifications and managing subscriptions

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

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

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)**.

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".

EXAMPLE 3: Alice's motorhome has a unique URI as id, but can be assigned multiple NGSI-LD Entity types, e.g. "Vehicle" and "Home".

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, whose 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 GeoProperty: subclass of NGSI-LD Property which is a 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, that uses the special *hasValue* property to define its target value and holds a geographic location in GeoJSON format

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

EXAMPLE: An NGSI-LD Entity, whose 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 LanguageProperty: subclass of NGSI-LD Property which is a description instance which associates a set of strings in different natural languages as a defined main characteristic, i.e. an **NGSI-LD Map**, to an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property and that uses the special *hasLanguageMap* (a subproperty of *hasValue*) property to define its target value

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 Map: JSON-LD language map in the form of key-value pairs holding the string representation of a main characteristic in a series of natural languages

EXAMPLE: "Bob's vehicle is currently parked on a street which is known as 'Grand Place' in French and 'Grote Markt' in Dutch" can be represented by an NGSI-LD LanguageProperty whose Name is "street" which holds an NGSI-LD Map of two key-value pairs containing both the French ("fr") and Dutch ("nl") exonyms of the street name.

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

NGSI-LD Null: *"urn:ngsi-ld:null"* or *{"@none": "urn:ngsi-ld:null"}* used as an encoding for *null* values

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

NGSI-LD Query: collection of criteria used to select a sub-set of NGSI-LD Entities, matching the criteria

NGSI-LD Registry API: part of the NGSI-LD API that is implemented by the Context Registry, including operations for registering Context Sources and managing Context Source Registrations (CSRs), operations for retrieving and discovering CSRs, and operations for subscribing to CSRs and receiving notifications

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

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

NGSI-LD Scope: enables putting Entities into a hierarchical structure and scoping queries and subscriptions according to it

NGSI-LD Temporal API: part of the NGSI-LD API pertaining to the Temporal Evolution of Entities, including operations for providing and managing the Temporal Evolution of Entities and Attributes, and operations for consuming the Temporal Evolution of Entities

NGSI-LD Temporal Evolution of Entities: sequence of values attributed to them over time, i.e. their history or future predictions

NGSI-LD Tenant: user or a group of users that utilize a single instance of a system implementing the NGSI-LD API (NGSI-LD Context Source or NGSI-LD Broker) in isolation from other users or groups of users of the same instance. Any information related to one tenant (e.g. Entities, Subscriptions, Context Source Registrations) are only visible to users of the same tenant, but not to users of a different tenant

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).

NGSI-LD VocabProperty: subclass of NGSI-LD Property which is a description instance which associates a string value which can be coerced to a URI as a defined main characteristic, i.e. an NGSI-LD Vocabulary, to an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property and that uses the special *hasVocab* (a subproperty of *hasValue*) property to define its target value

NGSI-LD Vocabulary: string representation of a main characteristic which is explicitly defined to undergo JSON-LD type coercion to a URI

EXAMPLE: "Bob's car is a non-commercial vehicle" can be represented by an NGSI-LD VocabProperty whose Name is "category" which holds an NGSI-LD Vocabulary with the string value *"non-commercial"*. If the associated JSON-LD context defines the term *"non-commercial"* as *"http://example.com/ non-commercial"*, then the returned value shall be the expanded using type coercion into the IRI the *http://example.com/ non-commercial*.

3.2 Symbols

Void.

3.3 Abbreviations

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

ABNF	Augmented Backus-Naur Form
ALG1	Algorithm for transforming an NGSI-LD Entity into a JSON-LD document
AM	Ante Meridiem
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BNF	Backus Naur Form
CH	Switzerland
CSR	Context Source Registration
ECMA	European Computer Manufacturers Association
EU	European Union
FI	Future Internet
FQN	Fully Qualified Name
GB	Great Britain
GDPR	General Data Protection Regulation
GeoJSON	Geographic JavaScript Object Notation
GeoJSON-LD	Geographic JavaScript Object Notation - Linked Data
GIS	Geographic Information System
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
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
LD	Linked Data
LWM2M	LightWeight Machine to Machine
M2M	Machine to Machine

MIME	Multi-purpose Internet Mail Extensions
MQTT	Message Queuing Telemetry Transport
N/A	Not Applicable
NGSI	Next Generation Service Interfaces
NGSILD	Next Generation Service Interfaces Linked Data (same as NGSI-LD)
NID	Namespace Identifier
NSS	Namespace Specific String
OAS	Open API Specification
OMA	Open Mobile Alliance
oneM2M	oneM2M Partnership Project
PM	Post Meridiem
POSIX	Portable Operating System Interface
QoS	Quality of Service
RDF	Resource Description Format
REST	Representational State Transfer
RFC	Request For Comments
SAREF	Smart Applications Reference ontology
TCP	Transport Control Protocol
TLS	Transport Layer Security
TS	Technical Specification
UCA	Unicode Collation Algorithm
UL	Ultra Light
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Universal Resource Locator
URN	Uniform Resource Name
UTC	Coordinated Universal Time
UTF	Unicode (or Universal Coded Character Set) Transformation Format
XSD	XML Schema Definition

4 Context Information Management Framework

4.1 Introduction

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

4.2 NGSI-LD Information Model

4.2.1 Introduction

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

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

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

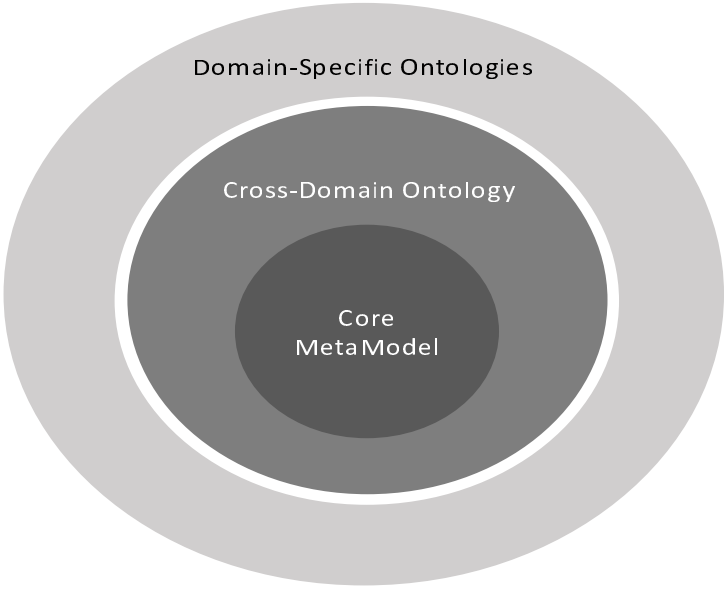

Figure 4.2.1-1: Overview of the NGSI-LD Information Model Structure

4.2.2 NGSI-LD Meta Model

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

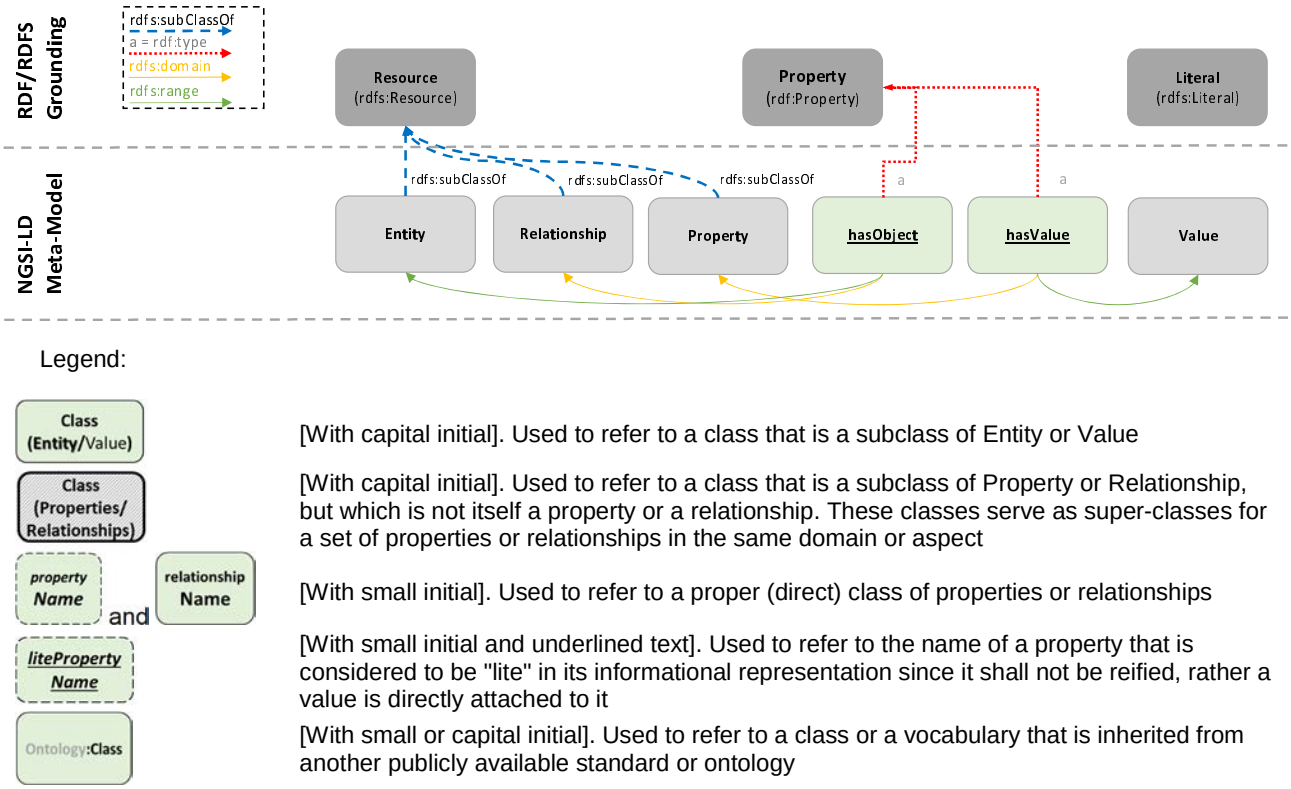

Figure 4.2.2-1: NGSI-LD Core Meta-Model

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

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

4.2.3 Cross Domain Ontology

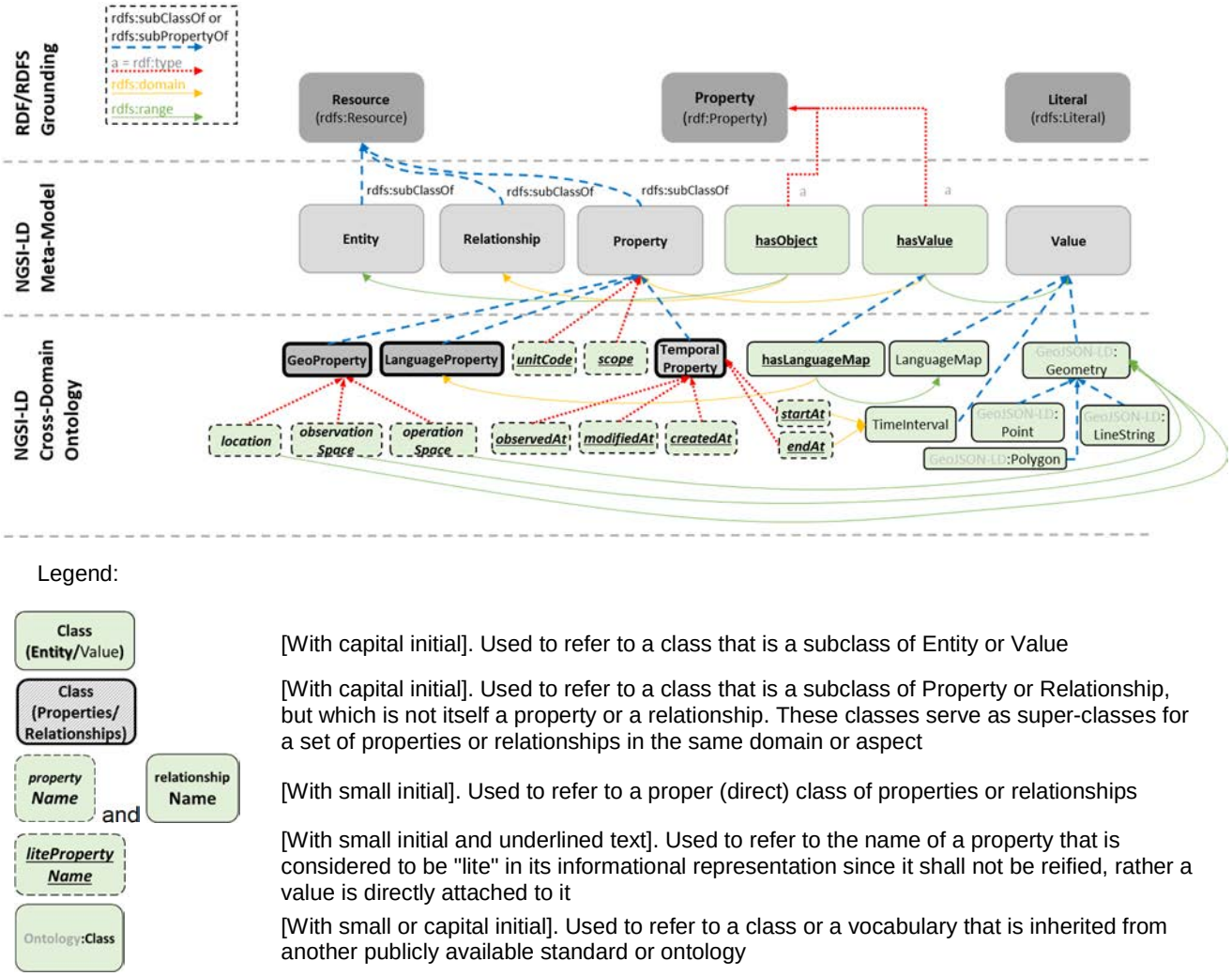

Figure 4.2.3-1: NGSI-LD Core Meta-Model plus the Cross-Domain Ontology

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

- **Geo Properties:** Are intended to convey geospatial information and implementations shall support them as defined in clause 4.7.
- **Temporal Properties:** Are non-reified Properties (represented only by their Value) that convey temporal information for capturing the time series evolution of other Properties; implementations shall support them as defined in clause 4.8.

- **Language Properties:** Are intended to convey different versions of the same textual values, whenever a version for each language (for instance: English, Spanish) is needed.
- **"unitCode" Property:** Is a Property intended to provide the units of measurement of an NGSI-LD Value. Implementations shall support it as defined in clause 4.5.2.
- **"scope" Property:** Is a Property that enables putting Entities into a hierarchical structure. Implementations shall support it as defined in clause 4.18.
- **LanguageMaps:** Are a special type of NGSI-LD Value intended to convey the different values of Language Properties, stated through an *hasLanguageMap*, which is of type rdf:Property [1] and is itself a subproperty of *hasValue*.
- **Geometry Values:** 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:** Are a special type of NGSI-LD Value intended to convey time instants or intervals representations. Implementations shall support them as defined in clause 4.6.3.

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

4.2.4 NGSI-LD domain-specific models and instantiation

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

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

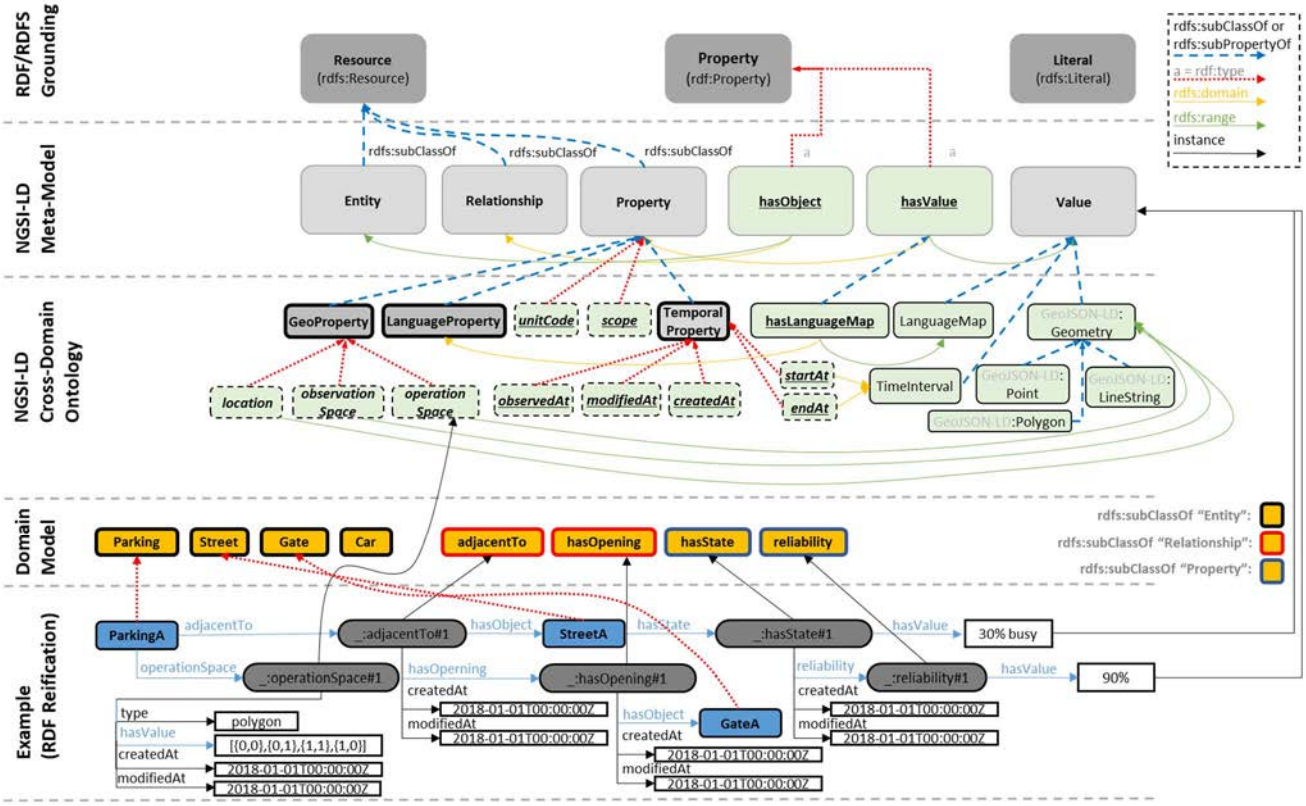

Figure 4.2.4-1: Cross-Domain Ontology and instantiation

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

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

4.2.5 UML representation

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

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

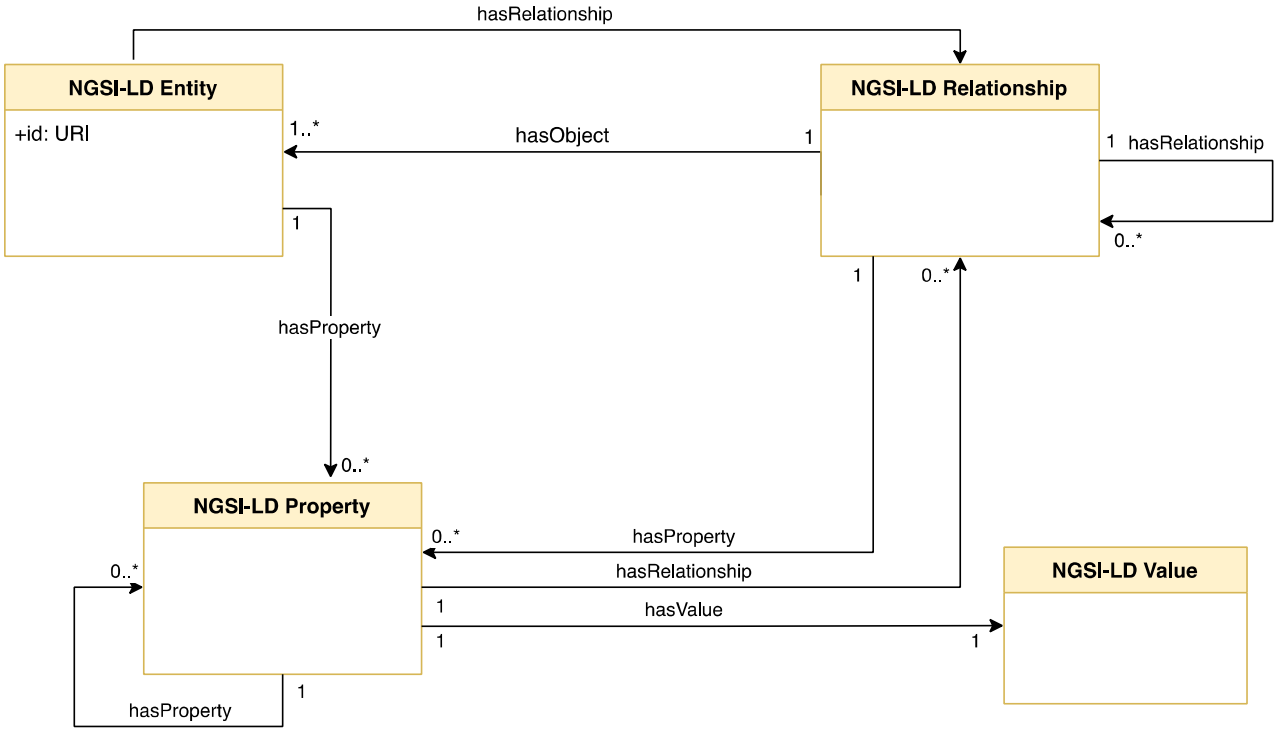

Figure 4.2.5-1: NGSI-LD information model as UML

4.3 NGSI-LD Architectural Considerations

4.3.1 Introduction

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

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

The NGSI-LD API implicitly defines two sets of Entities:

- the "current state";
- the "temporal evolution" (both the past and possibly future predictions).

The NGSI-LD API is structured into a Core API and an optional Temporal API. The Core API manages the current state of Entities. The Temporal API is optional and manages the Temporal Evolution of Entities. Brokers that intend to implement the Temporal API should consider updating the Temporal Evolution of an Entity whenever the "current state" is modified via the Core API.

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.

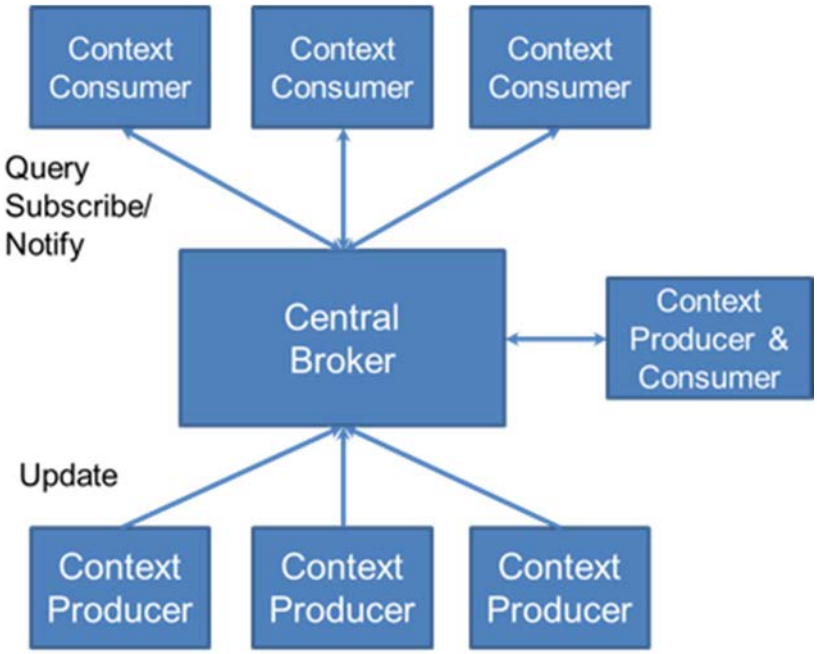

Figure 4.3.2-1: Centralized architecture

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.

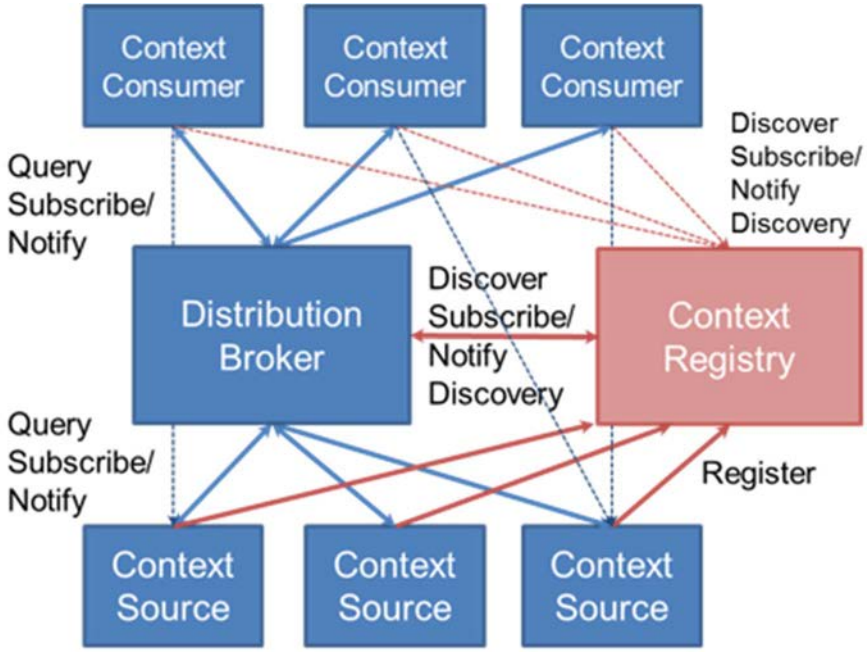

Figure 4.3.3-1: Distributed architecture

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

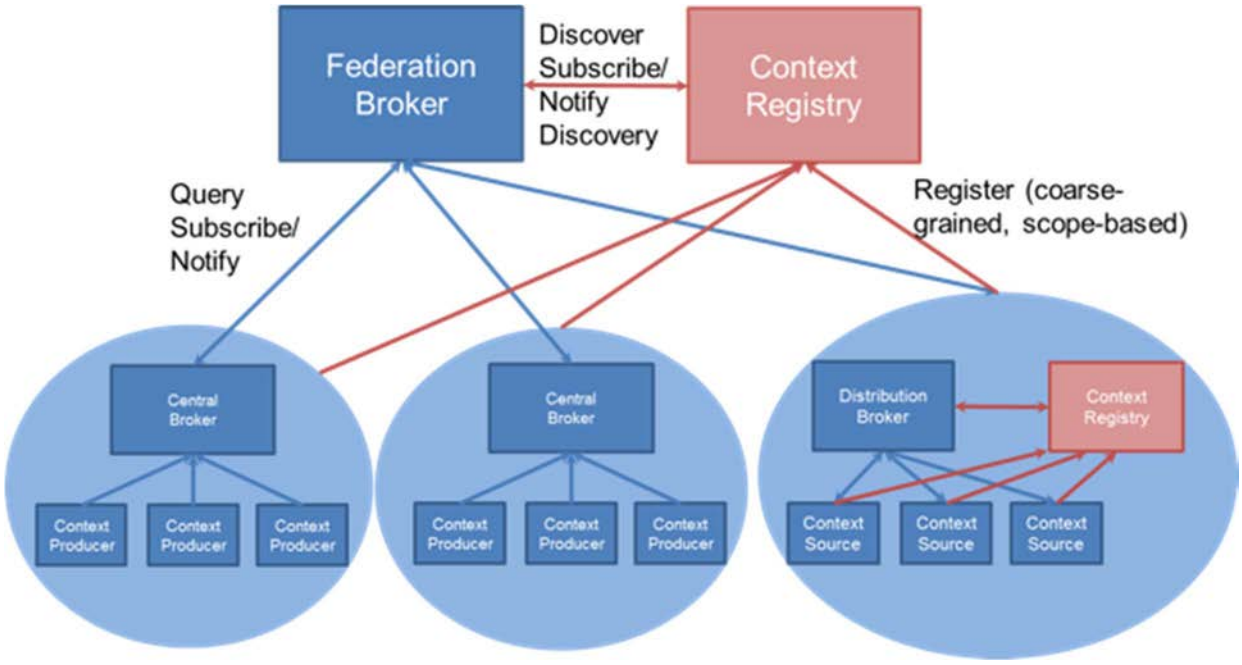

Figure 4.3.4-1: Federated architecture

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

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

4.3.5 NGSI-LD API Structure and Implementation Options

As stated in clause 4.3.1, the NGSI-LD API is structured into a Core API and an optional Temporal API. In addition, the Registry API consists of the operations to be implemented by the Context Registry. Furthermore, the JSON-LD Context API provides functionality for storing, managing and serving JSON-LD @contexts. The APIs are structured according to their functionalities, which is also reflected in how the operations are structured in clause 5. Table 4.3.5-1 introduces the API structure, the respective functionalities and lists the operations for each functionality, pointing to the clauses in which they are defined.

Table 4.3.5-1: NGSI-LD API structure

API	Functionality	Operations
Core API	Context Information Provision - operations for providing or managing Entities and Attributes	5.6.1 Create Entity 5.6.2 Update Attributes 5.6.3 Append Attributes 5.6.4 Partial Attribute Update 5.6.5 Delete Attribute 5.6.6 Delete Entity 5.6.7 Batch Entity Creation 5.6.8 Batch Entity Upsert 5.6.9 Batch Entity Update 5.6.10 Batch Entity Delete 5.6.17 Merge Entity 5.6.18 Replace Entity 5.6.19 Attribute replace 5.6.20 Batch Entity Merge
	Context Information Consumption - operations for consuming Entities and checking for which Entity Types and Attributes Entities are available in the system	5.7.1 Retrieve Entity 5.7.2 Query Entities 5.7.5 Retrieve Available Entity Types 5.7.6 Retrieve Details of Available Entity Types 5.7.7 Retrieve Available Entity Type Information 5.7.8 Retrieve Available Attributes 5.7.9 Retrieve Details of Available Attributes 5.7.10 Retrieve Available Attribute Information

API	Functionality	Operations
	Context Information Subscription - operations for subscribing to Entities, receiving notifications and managing subscriptions	5.8.1 Create Subscription 5.8.2 Update Subscription 5.8.3 Retrieve Subscription 5.8.4 Query Subscription 5.8.5 Delete Subscription 5.8.6 Notification
Temporal API	Temporal Context Information Provision - operations for providing or managing the Temporal Evolution of Entities and Attributes	5.6.11 Upsert Temporal Representation 5.6.12 Add Attributes to Temporal Representation 5.6.13 Delete Attributes from Temporal Representation 5.6.14 Partial Update Attribute instance 5.6.15 Delete Attribute Instance 5.6.16 Delete Temporal Representation
	Temporal Context Information Consumption - operations for consuming the Temporal Evolution of Entities	5.7.3 Retrieve Temporal Evolution of Entity 5.7.4 Query Temporal Evolution of Entities
Registry API	Context Source Registration - operations for registering Context Sources and managing Context Source Registrations (CSRs)	5.9.2 Register Context Source 5.9.3 Update CSR 5.9.4 Delete CSR
	Context Source Discovery - operations for retrieving and discovering CSRs	5.7.1 Retrieve CSR 5.7.2 Query CSRs
	Context Source Registration Subscription - operations for subscribing to CSRs, receiving notifications and managing CSRs	5.11.2 Create CSR Subscription 5.11.3 Update CSR Subscription 5.11.4 Retrieve CSR Subscription 5.11.5 Query CSR Subscription 5.11.6 Delete CSR Subscription 5.11.7 CSR Notification
JSON-LD Context API	Storing, managing and serving @contexts	5.13.2 Add @context 5.13.3 List @contexts 5.13.4 Serve @context 5.13.5 Delete and Reload @context

All Brokers shall implement the Core API. Temporal API and Registry API can be implemented by a Broker or by a separate temporal component and Context Registry respectively. Table 4.3.5-2 shows the possible implementation configurations. A temporal component implementing the Temporal API can also be used completely independently of a Broker. The JSON-LD Context API is optional. The managing and serving of @contexts can also be handled by an independent, stand-alone component.

Table 4.3.5-2: Main implementation configurations

Description	Temporal API	Registry API
Central Broker without temporal support	none	none
Central Broker with integrated temporal component	local	none
Central Broker with separate temporal component	separate	none
Broker supporting distributed and federated deployments without temporal support and with integrated Context Registry	none	local
Broker supporting distributed and federated deployments with integrated temporal component and integrated Context Registry	local	local
Broker supporting distributed and federated deployments with separate temporal component and integrated Context Registry	separate	local
Broker supporting distributed and federated deployments without temporal support and separate Context Registry	none	separate
Broker supporting distributed and federated deployments with integrated temporal component and separate Context Registry	local	separate
Broker supporting distributed and federated deployments with separate temporal component and separate Context Registry	separate	separate

Table 4.3.5-3 shows which operations are implemented and used by the other architectural roles as introduced in clause 4.3.2, clause 4.3.3 and clause 4.3.4. In addition, there are separate roles for the temporal API, i.e. Temporal Context Producer, Temporal Context Source and Temporal Context Consumer. For completeness, the roles of Context Repository and Temporal Context Repository have been introduced, implementing the Context Information Provision and Temporal Context Information Provision functionalities, respectively. In practice, components implementing the latter roles will also implement functionalities for consuming or processing the stored information. Actual components can have multiple roles at the same time, e.g. a Broker can implement all roles at the same time. Context Consumers typically only interact with Brokers, but in alternative setups, as shown in figure 4.3.3-1, they can also directly interact with the Context Registry and then directly contact Context Sources.

Table 4.3.5-3: Operations implemented by the various NGSI-LD Roles

NGSI-LD Role	Implements	Uses
Context Consumer	5.8.6 Notification - *if supporting asynchronous interactions* In case of direct interactions with Context Registry: 5.11.7 CSR Notification - *if supporting asynchronous interactions*	5.7.1 Retrieve Entity 5.7.2 Query Entities 5.7.5 Retrieve Available Entity Types 5.7.6 Retrieve Details of Available Entity Types 5.7.7 Retrieve Available Entity Type Information 5.7.8 Retrieve Available Attributes 5.7.9 Retrieve Details of Available Attributes 5.7.10 Retrieve Available Attribute Information 5.8.1 Create Subscription 5.8.2 Update Subscription 5.8.3 Retrieve Subscription 5.8.4 Query Subscription 5.8.5 Delete Subscription In case of direct interactions with Context Registry: 5.7.1 Retrieve CSR 5.7.2 Query CSRs *… if supporting asynchronous interactions* 5.11.2 Create CSR Subscription 5.11.3 Update CSR Subscription 5.11.4 Retrieve CSR Subscription 5.11.5 Query CSR Subscription 5.11.6 Delete CSR Subscription
Context Producer	none	5.6.1 Create Entity 5.6.2 Update Attributes 5.6.3 Append Attributes 5.6.4 Partial Attribute Update 5.6.5 Delete Attribute 5.6.6 Delete Entity 5.6.7 Batch Entity Creation 5.6.8 Batch Entity Upsert 5.6.9 Batch Entity Update 5.6.10 Batch Entity Delete 5.6.17 Merge Entity 5.6.18 Replace Entity 5.6.19 Attribute replace 5.6.20 Batch Entity Merge
Context Source	5.7.1 Retrieve Entity 5.7.2 Query Entities 5.7.5 Retrieve Available Entity Types 5.7.6 Retrieve Details of Available Entity Types 5.7.7 Retrieve Available Entity Type Information 5.7.8 Retrieve Available Attributes 5.7.9 Retrieve Details of Available Attributes 5.7.10 Retrieve Available Attribute Information 5.8.1 Create Subscription 5.8.2 Update Subscription 5.8.3 Retrieve Subscription 5.8.4 Query Subscription 5.8.5 Delete Subscription	5.8.6 Notification 5.9.2 Register Context Source 5.9.3 Update CSR 5.9.4 Delete CSR

NGSI-LD Role	Implements	Uses
Context Repository	5.6.1 Create Entity 5.6.2 Update Attributes 5.6.3 Append Attributes 5.6.4 Partial Attribute Update 5.6.5 Delete Attribute 5.6.6 Delete Entity 5.6.7 Batch Entity Creation 5.6.8 Batch Entity Upsert 5.6.9 Batch Entity Update 5.6.10 Batch Entity Delete 5.6.17 Merge Entity 5.6.18 Replace Entity 5.6.19 Attribute replace 5.6.20 Batch Entity Merge	none
Temporal Context Consumer	In case of direct interactions with Context Registry: 5.11.7 CSR Notification - *if supporting asynchronous interactions*	5.7.3 Retrieve Temporal Evolution of Entity 5.7.4 Query Temporal Evolution of Entities In case of direct interactions with Context Registry: 5.7.1 Retrieve CSR 5.7.2 Query CSRs *… if supporting asynchronous interactions* 5.11.2 Create CSR Subscription 5.11.3 Update CSR Subscription 5.11.4 Retrieve CSR Subscription 5.11.5 Query CSR Subscription 5.11.6 Delete CSR Subscription
Temporal Context Producer	None	5.6.11 Upsert Temporal Representation 5.6.12 Add Attributes to Temporal Representation 5.6.13 Delete Attributes from Temporal Representation 5.6.14 Partial Update Attribute instance 5.6.15 Delete Attribute Instance 5.6.16 Delete Temporal Representation
Temporal Context Source	5.7.3 Retrieve Temporal Evolution of Entity 5.7.4 Query Temporal Evolution of Entities	5.9.2 Register Context Source 5.9.3 Update CSR 5.9.4 Delete CSR
Temporal Context Repository	5.6.11 Upsert Temporal Representation 5.6.12 Add Attributes to Temporal Representation 5.6.13 Delete Attributes from Temporal Representation 5.6.14 Partial Update Attribute instance 5.6.15 Delete Attribute Instance 5.6.16 Delete Temporal Representation	none
Context Registry	5.9.2 Register Context Source 5.9.3 Update CSR 5.9.4 Delete CSR 5.7.1 Retrieve CSR 5.7.2 Query CSRs 5.11.2 Create CSR Subscription 5.11.3 Update CSR Subscription 5.11.4 Retrieve CSR Subscription 5.11.5 Query CSR Subscription 5.11.6 Delete CSR Subscription	5.11.7 CSR Notification

4.3.6 Distributed Operations

4.3.6.1 Introduction

One fundamental concept underpinning all of the prototypical architectures described above (clauses 4.3.2, 4.3.3 and 4.3.4) is the idea that Entity data does not need to be centralized within a single *Context Broker*. When reading context information, a *Context Broker* can be used as a single point of access to retrieve Entity data found distributed across multiple associated *Context Brokers* each receiving a *Context Consumption* request. Similarly, when modifying an Entity, a single request to a *Context Broker* may result in the operation being distributed and different parts of that Entity being updated across multiple *Context Brokers* each receiving a *Context Provision* request.

As long as there is only a centralized *Context Broker*, i.e. there are no *Context Sources* registered, all NGSI-LD requests, with the exception of *Update Attributes* (see clause 5.6.2) and the batch operations (see clauses 5.6.7, 5.6.8, 5.6.9, 5.6.10, 5.6.20), can either be successfully executed completely, or result in an error. In the distributed case, all requests can be partially successful. For the centralized case described above, only *Update Attributes* and the batch operations can be partially successful.

When a Context Source is registered, an operation mode is selected. This defines the basis for distributed operations and also defines whether or not the *Context Broker* is permitted to hold context data about the Entities and Attributes locally itself.

If two registered Context Sources are providing context data for the same Attribute, the Attribute instances can be distinguished by datasetId. The mechanism for determining which data shall be returned is defined in clause 4.5.5.

It is possible to restrict a registered Context Source to operate on a specific Entity type or list of Entity types. In order for Context Broker hierarchies to support and restrict the distribution of such limited operations, the Entity type selector (see clause 4.17) can be added as a filter on forwarded requests even where its presence initially seems redundant.

Furthermore, registered Context Sources may indicate that they are only willing to respond to a limited subset of API operations. *Context Brokers* shall respect this, to avoid unnecessarily sending distributed operation requests which are always guaranteed to fail. For example, a Context Source may consistently refuse certain API operations since it does not support them. Alternatively, some Context Source endpoints (such as updates) may be protected for use by authorized users only, and not accessible to a *Context Broker* without those rights. Limited access is likely to be the case in extended data sharing scenarios, where a registered Context Source, and the data held within it, may belong to an external third party.

For the endpoints served, all registered Context Sources shall support the normalized representation of Entities as default. Support of additional representation formats is optional and will depend on the implementation. System generated attributes such as *modifiedAt* and *createdAt* (see clause 4.8) should be supported by registered Context Sources, at a minimum no error shall be returned if they are not available when requested.

4.3.6.2 Additive Registrations

For additive registrations, the *Context Broker* is permitted to hold context data about the Entities and Attributes locally itself, and also obtain data from external sources. Context producing operations are serviced both locally by the *Context Broker* itself, and also distributed on to the registered sources.

An **inclusive** Context Source Registration specifies that the *Context Broker* considers all registered Context Sources as equals and will distribute operations to those Context Sources even if relevant context data is available directly within the Context Broker itself (in which case, all results will be integrated in the final response). Data from every Context Source registered by an inclusive Context Source Registration is requested with an equal priority. This is the default mode of operation.

An **auxiliary** Context Source Registration never overrides data held directly within a *Context Broker*. Auxiliary distributed operations are limited to context information consumption operations (see clause 5.7). Context data from auxiliary context sources is only included if it is supplementary to the context data otherwise available to the *Context Broker*. Auxiliary Context Source Registrations are always accepted as there can never be a conflict.

4.3.6.3 Proxied Registrations

For proxied registrations, the *Context Broker* itself is not permitted to hold context data about the registered Entities and Attributes locally (thus all registered context data is obtained from the external registered sources). Unregistered Attributes of an Entity are permitted to be held locally; when Context producing operations are received, registered Attributes are distributed on to the registered sources and never serviced directly by the *Context Broker* itself.

An **exclusive** Context Source Registration specifies that all of the registered context data is held in a single location external to the *Context Broker*. The *Context Broker* itself holds no data locally about the registered Attributes and no overlapping proxied Context Source Registrations shall be supported for the same combination of registered Attributes on the Entity. An **exclusive** registration shall be fully specified and always relates to specific Attributes found on a single Entity. Thus, the registration shall define **both:**

- An entity id (i.e. an id pattern or Entity type defining a group of entities is not supported for **exclusive** registrations).
- Attributes.

Once an **exclusive** Context Source Registration has been created, no further exclusive or redirect Context Source Registrations can be created for that same combination of Entity id and Attributes.

A **redirect** Context Source Registration also specifies that the registered context data is held in a location external to the *Context Broker*. It is possible to register (any combination of):

- A whole Entity by id or id pattern (i.e. without specifying individual Attributes in the registration; in this case, all Attributes are held externally).
- Entities by Entity type only (with or without specifying individual Attributes).
- Attributes only.

Potentially multiple distinct **redirect** registrations can apply at the same time. The *Context Broker* itself holds no data locally in conflict to the registration. In the case that multiple overlapping **redirect** registrations are defined, operations are distributed to all registered sources.

4.3.6.4 Limiting Cascading Distributed Operations

When creating a registration, it is unknown whether the requested data is held at the distributed endpoint, or it is in turn distributed via further registrations. It is necessary to include a binding-specific mechanism to request operations only on the registered endpoint itself to avoid cascades of an excessive lengths, duplicates or loops.

4.3.6.5 Extra information to provide when contacting Context Source

If the optional array (of *KeyValuePair* type, as defined by clause 5.2.22) "contextSourceInfo" of the CSourceRegistration is present, it contains, whatever extra information the Broker shall convey when contacting the Context Source. This can be information the Broker needs to successfully communicate with the Context Source (e.g. Authorization material), or for the Context Source to correctly interpret the received content (e.g. the Link URL to fetch an @context). The method for conveying this information is binding-specific, e.g. using headers in the case of HTTP.

Instead of providing the actual value, the special value "urn:ngsi-ld:request" can be used to indicate that the respective value is to be taken from the request that triggered the given request, if present.

EXAMPLE: If the key value pair "user":"urn:ngsi-ld:request" is part of "contextSourceInfo" of the CSourceRegistration, the Broker checks if "user" was conveyed in the triggering request. If this is the case, e.g. "user":"abcd", "user":"abcd" is also conveyed when contacting the Context Source.

As tenant information, if applicable, is directly specified in the CSourceRegistration, it shall not be part of "contextSourceInfo". Binding-specific information that is used for setting up the connection or is specific for an interaction, e.g. Content-length in HTTP, cannot be overridden by "contextSourceInfo". If present, such information shall be ignored.

4.3.6.6 Additional pre- and post-processing of extra information when contacting Context Source

The following key-values have a specific well-defined meaning when defined as elements within the optional array "contextSourceInfo" of the CSourceRegistration.

- If the key "jsonldContext" is defined, the value shall correspond to a URL reference as defined by the JSON-LD specification [2], section 3.1.
- The Context Broker shall apply a compaction operation as defined by the JSON-LD specification [2], section 4.1.5 over both payload and query parameters using the JSON-LD Context supplied in the value of the "jsonldContext" key-value pair, prior to distributing the request to the context source endpoint and forwarding with this JSON-LD context using an appropriate binding. Additionally, if a payload is defined in the initial request to the Context Broker, the "Content-Type" of the forwarded request shall be "application/json" and the Context Broker shall remove any "@context" members from the payload prior to distributing the request to the context source endpoint.
- If the key "accept" is defined, the value shall be a MIME type acceptable to the Context Broker (one of: "application/json", "application/ld+json").
- The response from the distributed endpoint shall be returned in this defined format and if necessary, the Context Broker shall be responsible for converting this to the desired content type when aggregating responses to the initial request.

4.4 Core and user NGSI-LD @context

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

- The core terms needed to uniquely represent the key concepts defined by the NGSI-LD Information Model, as mandated by clause 4.2.
- The terms needed to uniquely represent all the members that define the API-related Data Types, as mandated by clauses 5.2 and 5.3.
- A fallback *@vocab* rule to expand or compact user-defined terms to a default URI, in case there is no other possible expansion or compaction as per the current @context.
- The core NGSI-LD @context defines the term "id", which is mapped to "@id", and term "type", which is mapped to "@type". Since @id and @type are what is typically used in JSON-LD, they may also be used in NGSI-LD requests instead of "id" and "type" respectively, wherever this is applicable. In NGSI-LD responses, only "id" and "type" shall be used.

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

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

The NGSI-LD Core @context is publicly available at https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld and shall contain all the terms as mandated by annex B.

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

- One term associated to the Entity Type, mapping the Entity Type Name with its Type Identifier (URI).
- One term associated to the name of each Property or any of its subclasses 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 mapping the Relationship Name with the Relationship Identifier (URI) in the form:
  - "@type": "@id".
  - "@id": <Relationship's URI>.

The user @context shall not contain JSON-LD Scoped Contexts (see [2], section 4.1.8), as described in clause 5.5.7.

Depending on the binding, the @context may not just be provided embedded with the rest of the JSON content, but there could be other options. For example, in the HTTP binding, the @context can be made available through a Link header (see clause 6.3.5).

4.5 NGSI-LD Data Representation

4.5.0 Introduction

All NGSI-LD elements are represented in JSON-LD [2]. For the use with the API, the compacted JSON-LD representation is used, i.e. short terms are used, which are expanded by the component implementing the NGSI-LD API using a JSON-LD @context, typically provided as part of the request. As described in clause 4.4, the NGSI-LD Core @context is always considered to be part of the @context to be used.

The use of JSON-LD for NGSI-LD elements has some implications for the use of *null* values, as JSON-LD interprets setting elements to *null* as elements to be removed when performing JSON-LD expansion. Thus, *null* cannot be used as a value in NGSI-LD.

To nevertheless allow deletions as part of NGSI-LD operations that update NGSI-LD data, which is typically handled by setting the respective JSON key to *null* (e.g. as in IETF RFC 7396 [16]), the URI *"urn:ngsi-ld:null"* is used as a replacement for *null* in all places, where URI strings are valid JSON values. For *languageMap*, the JSON object *{"@none": "urn:ngsi-ld:null"}* is to be used as explained in clause 4.5.18. These encodings of *null* are referred to as NGSI-LD Null.

For representing deleted elements in notifications and in the temporal evolution, the URI *"urn:ngsi-ld:null"* is used as a Property *value* or Relationship *object* and the JSON object *{"@none": "urn:ngsi-ld:null"}* for the *languageMap* of a Language Property, respectively.

As *null* cannot be used as a value in JSON-LD, there is still the possibility of using a JSON null literal represented as *{"@type": "@json", "@value": null}* in JSON-LD instead. JSON literals are not to be expanded in JSON-LD and thus the respective element is not removed during JSON-LD expansion.

4.5.1 NGSI-LD Entity Representation

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

The JSON-LD object contains the following members:

- "id" whose value shall be a URI that identifies the Entity. *Mandatory*.

- "type" whose value shall be equal to the Entity Type Name or an unordered JSON array with multiple Entity Type Names in case of an Entity that has multiple Entity Types. *Mandatory*.
- "scope" whose value shall be a Scope as defined in clause 4.18 or an unordered JSON array with multiple Scopes in case of an Entity that has multiple Scopes. *Optional*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- One member for each Property as per the rules stated in clause 4.5.2. In case of multiple Property instances with the same Property Name as described in clause 4.5.5, all instances are provided as an unordered JSON array.
- One member for each Relationship as per the rules stated in clause 4.5.3. In case of multiple Relationship instances with the same Property Name as described in clause 4.5.5, all instances are provided as an unordered JSON array.

NOTE 1: In the following, the term Attribute is used when referring in the text to both a Property and a Relationship (see definition of NGSI-LD Attribute in clause 3.1).

NOTE 2: When GeoJSON representation is selected, the layout of the Entities changes, see clause 4.5.16 for details.

Terms defined in the Core Context as non-reified Properties (such as "datasetId", "instanceId", etc.) shall not be used as Attribute names.

Attributes shall not contain any embedded @context, as described in clause 5.5.7.

4.5.2 NGSI-LD Property Representations

4.5.2.1 Introduction

An NGSI-LD Property, its value and sub-attributes can be represented in two equally valid lossless formats. The **normalized representation** is a JSON-LD document that is complete with respect to mandatory members. The **concise representation** is a terser alternative, which makes various implicit assumptions against the payloads and removes redundancy from them.

Both normalized and concise representation of Properties shall be supported by implementations and can be selected by Context Consumers through specific request parameters. An example of this representation can be found in annex C, clause C.2.2.

4.5.2.2 Normalized NGSI-LD Property

An NGSI-LD Property in normalized representation shall be represented by a member whose key is the Property Name (a term) and whose value is a JSON-LD object (or JSON-LD array with such JSON-LD objects, if there are multiple instances with the same Property Name, as described in clause 4.5.5), which includes the following members:

- "type": "Property". *Mandatory*.
- "value": the Property Value (see definition of NGSI-LD Value 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, else the member's value shall be a JSON object in the form:
  - "@type": <Data Type URI>.
  - "@value": Property Value.

  An NGSI-LD Null (explained in clause 4.5.0 and defined in clause 3.1) can be used as the right-hand side of the "value" during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) to indicate a deletion of the Property, as well as in notifications and in temporal evolution (for encoding a deleted Property).
- "previousValue": only provided in case of notifications and if the *showChanges* option is explicitly requested. It represents the previous Property Value, before the triggering change. The representation is the same as that of "value". *Optional*.

- "observedAt": a string as mandated by clause 4.8. *Optional*.
- "datasetId": a URI as mandated by clause 4.5.5. *Optional.*
- "instanceId": a URI uniquely identifying a Property instance, as mandated by (see clause 4.5.7). *System generated. Optional.*
- "createdAt": a string as mandated by clause 4.8. *System generated.*
- "modifiedAt": a string as mandated by clause 4.8. *System generated.*
- "deletedAt": 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 UNECE/CEFACT Common Codes for Units of Measurement [15]. *Optional*.
- "object" and "previousObject": shall never be present, as they define a Relationship's object URI. *Prohibited*.
- "languageMap" and "previousLanguageMap": shall never be present, as they define a LanguageProperty value. *Prohibited*.
- "vocab" and "previousVocab": shall never be present, as they define a VocabularyProperty value. *Prohibited*.
- For each of the Properties this Property is associated with, a member whose key (a term) is the Property Name and value is the result of serializing a **Property** (or any of its subclasses) in normalized representation (see clause 4.5.2.2).
- For each of the Relationships this Property is associated with, a member whose key (a term) is the Relationship Name and value is the result of serializing a **Relationship** in normalized representation (see clause 4.5.3.2).

4.5.2.3 Concise NGSI-LD Property

An NGSI-LD Property without sub-attributes shall be represented in a concise but lossless representation by a member whose key is the Property Name (a term) and whose value is the Property Value (see definition of NGSI-LD Value in clause 3.1). In this case the concise representation is equivalent to simplified representation (see clause 4.5.4).

- "type": shall never be present, as "Property" can be inferred. An exception to this inference rule occurs for geospatial Property Values, where the "GeoProperty" sub-type shall be inferred instead, if the Property Value resolves to a supported GeoJSON geometry (see clause 4.7). *Prohibited*.
- "value": shall never be present, as it can be inferred. *Prohibited.*

During partial update patch and merge patch (see clauses 5.5.8 and 5.5.12), when deleting a Property without a *datasetId*, as well as when notifying about a deleted Property without sub-attributes, the NGSI-LD Property should be represented in a concise representation by a member whose key is the Property Name (a term) and whose value is *"urn:ngsi-ld:null"*.

An NGSI-LD Property which includes additional sub-attributes shall be represented in a concise but lossless representation by a member whose key is the Property Name (a term) and whose value is a JSON-LD object (or JSON-LD array with such JSON-LD objects if there are multiple instances with the same Property Name as described in clause 4.5.5) including the following members:

- "type": *Optional*. If missing, "Property" can be inferred by the presence of the "value" attribute. An exception to this inference rule occurs for geospatial Property Values, where the "GeoProperty" sub-type shall be inferred instead, if the Property Value resolves to a supported GeoJSON geometry (see clause 4.7).
- "value": the Property Value (see definition of NGSI-LD Value 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, else the member's value shall be a JSON object in the form:
  - "@type": <Data Type URI>.
  - "@value": Property Value.

An NGSI-LD Null (explained in clause 4.5.0 and defined in clause 3.1) can be used as the right-hand side of the "value" during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) to indicate a deletion of the Property, as well as in notifications and in temporal evolution (for encoding a deleted Property).

- "previousValue": only provided in case of notifications and if the *showChanges* option is explicitly requested. It represents the previous Property Value, before the triggering change. The representation is the same as that of "value". *Optional*.
- "observedAt": a string as mandated by clause 4.8. *Optional*.
- "datasetId": a URI as mandated by clause 4.5.5. *Optional.*
- "instanceId": a URI uniquely identifying a Property instance (see clause 4.5.7). *System generated. Optional.*
- "createdAt": a string as mandated by clause 4.8. *System generated.*
- "modifiedAt": a string as mandated by clause 4.8. *System generated.*
- "deletedAt": 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 UNECE/CEFACT Common Codes for Units of Measurement [15]. *Optional*.
- "object" and "previousObject": shall never be present, as they define a Relationship's object URI. *Prohibited*.
- "languageMap" and "previousLanguageMap": shall never be present, as they define a LanguageProperty value. *Prohibited*.
- "vocab" and "previousVocab": shall never be present, as they define a VocabularyProperty value. *Prohibited*.
- For each of the Properties this Property is associated with, a member whose key (a term) is the Property Name and value is the result of serializing a **Property** (or any of its subclasses) in concise representation (see clause 4.5.2.3).
- For each of the Relationships this Property is associated with, a member whose key (a term) is the Relationship Name and value is the result of serializing a **Relationship** in concise representation (see clause 4.5.3.3).

4.5.3 NGSI-LD Relationship Representations

4.5.3.1 Introduction

An NGSI-LD Relationship, its value and sub-attributes can be represented in two equally valid lossless formats. The **normalized representation** is a JSON-LD document that is complete with respect to mandatory members. The **concise representation** is a terser alternative, which makes various implicit assumptions against the payloads and removes redundancy from them.

Both normalized and concise representation of Relationships shall be supported by implementations and can be selected by Context Consumers through specific request parameters. An example of this representation can be found in annex C, clause C.2.2.

4.5.3.2 Normalized NGSI-LD Relationship

An NGSI-LD Relationship in normalized representation shall be represented by a member whose key is the Relationship Name (a term) and whose value is a JSON-LD object (or JSON-LD array with such JSON-LD objects, if there are multiple instances with the same Relationship Name, as described in clause 4.5.5) with the following terms:

- "type": "Relationship". *Mandatory.*
- "object": the Relationship's object represented by a URI. *Mandatory*.
  An NGSI-LD Null (explained in clause 4.5.0 and defined in clause 3.1) can be used as the right-hand side of the "object" during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) to indicate a deletion of the Relationship, as well as in notifications and in temporal evolution (for encoding a deleted Relationship).

- "previousObject": only provided in case of notifications and if the *showChanges* option is explicitly requested. It represents the previous Relationship *object*, before the triggering change. The representation is the same as that of "object". *Optional*.
- "observedAt": a string as mandated by clause 4.8. *Optional*.
- "datasetId": a URI as mandated by clause 4.5.5. *Optional.*
- "instanceId": a URI uniquely identifying a Relationship instance (see clause 4.5.8). *System generated. Optional.*
- "createdAt": a string as mandated by clause 4.8. *System generated*.
- "modifiedAt": a string as mandated by clause 4.8. *System generated*.
- "deletedAt": a string as mandated by clause 4.8. *System generated.*
- "value" and "previousValue": shall never be present, as they define a Property value. *Prohibited*.
- "languageMap" and "previousLanguageMap": shall never be present, as they define a LanguageProperty value. *Prohibited*.
- "vocab" and "previousVocab": shall never be present, as they define a VocabularyProperty value. *Prohibited*.
- "unitCode": shall never be present, as Relationships are unitless. *Prohibited*.
- For each Relationship 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** in normalized representation (see clause 4.5.3.2).
- For each Property 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** in normalized representation (see clause 4.5.2.2).

4.5.3.3 Concise NGSI-LD Relationship

An NGSI-LD Relationship in shall be represented in a concise but lossless representation by a member whose key is the Relationship Name (a term) and whose value is a JSON-LD object (or JSON-LD array with such JSON-LD objects if there are multiple instances with the same Relationship Name as described in clause 4.5.5) with the following terms:

- "type": *Optional*. If missing, "Relationship" can be inferred by the presence of the "object" attribute.
- "object": the Relationship's object represented by a URI. *Mandatory*.
  An NGSI-LD Null (explained in clause 4.5.0 and defined in clause 3.1) can be used as the right-hand side of the "object" during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) to indicate a deletion of the Relationship, as well as in notifications and in temporal evolution (for encoding a deleted Relationship).
- "previousObject": only provided in case of notifications and if the *showChanges* option is explicitly requested. It represents the previous Relationship *object*, before the triggering change. The representation is the same as that of "object". *Optional*.
- "observedAt": a string as mandated by clause 4.8. *Optional*.
- "datasetId": a URI as mandated by clause 4.5.5. *Optional.*
- "instanceId": a URI uniquely identifying a Relationship instance. *System generated. Optional.*
- "createdAt": a string as mandated by clause 4.8. *System generated.*
- "modifiedAt": a string as mandated by clause 4.8. *System generated*.
- "deletedAt": a string as mandated by clause 4.8. *System generated.*
- "value" and "previousValue": shall never be present, as they define a Property value. *Prohibited*.

- "languageMap" and "previousLanguageMap": shall never be present, as they define a LanguageProperty value. *Prohibited*.
- "vocab" and "previousVocab": shall never be present, as they define a VocabularyProperty value. *Prohibited*.
- "unitCode": shall never be present, as Relationships are unitless. *Prohibited*.
- For each Relationship 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** in concise representation (see clause 4.5.3.3).
- For each Property 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** in concise representation (see clause 4.5.2.3).

Notwithstanding the definition above, during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12), an NGSI-LD Relationship with a value of NGSI-LD Null and without a *datasetId* should be represented in a concise representation by a member whose key is the Relationship name (a term) and whose value is *"urn:ngsi-ld:null"*.

4.5.4 Simplified Representation

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

The simplified representation of an entity shall be a JSON-LD object containing the following members:

- "id" whose value shall be a URI that identifies the Entity. *Mandatory*.
- "type" whose value shall be equal to the Entity Type Name or an unordered JSON array with multiple Entity Type Names in case of an Entity that has multiple Entity Types. *Mandatory*.
- "@context", a JSON-LD @context as described in clause 4.4. Optional.
- For each Property a member whose key is the Property Name (a term) and whose value is the Property Value.

EXAMPLE 1: `"name": "David Robert Jones"`

- In the multi-attribute case (see clause 4.5.5), the simplified representation of a Property (or any of its subtypes) changes. Each Property consists of a key-value pair, the key being the Property Name (a term) and the value being a JSON Object, which contains a single Attribute with a key called "dataset", and its value in turn is a JSON Object holding a series of key-value pairs, one for each *datasetId*, where the value corresponds to the simplified representation of the property value. The default *datasetId* (where present) is represented by the JSON-LD keyword *@none*.

EXAMPLE 2:

```
"name": {
  "dataset": {
    "@none": "David Robert Jones",
    "urn:ngsi-ld:datasetId:001": "David Bowie",
    "urn:ngsi-ld:datasetId:002": "Ziggy Stardust"
  }
}
```

- For each GeoProperty, a member whose key is the Property Name (a term) and whose value is the Property Value.

EXAMPLE 3: `"location": {"type": "Point", "coordinates": [13.3986, 52.5547]}`

- For each LanguageProperty, a member whose key is the Property Name (a term) and whose value is a JSON Object containing a single Attribute with a key called "languageMap" where the value shall correspond to a LanguageProperty languageMap.

EXAMPLE 4: `"says": {"languageMap": {"en": "yes", "fr": "oui"}`

- For each VocabProperty, a member whose key is the Property Name (a term) and whose value is a JSON Object containing a single Attribute with a key called "vocab" where the value shall correspond to a VocabProperty vocab.

EXAMPLE 5: `"gender": {"vocab": "Male"}`

- 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)

EXAMPLE 6: `"knows": "urn:ngsi-ld:Person:31415`

- In the multi-attribute case (see clause 4.5.5), the simplified representation of a Relationship changes. Each Relationship consists of a key-value pair, the key being the Relationship Name (a term) and the value being a JSON Object containing a single Attribute with a key called "dataset" and its value in turn is a JSON Object holding a series of key-value pairs, one for each *datasetId* where the value corresponds to the object of the relationship. The default *datasetId* (where present) is represented by the JSON-LD keyword *@none*.

EXAMPLE 7:

```
"knows": {
  "dataset": {
    "@none": "urn:ngsi-ld:Person:31415",
    "urn:ngsi-ld:datasetId:001": "urn:ngsi-ld:Person:27182",
    "urn:ngsi-ld:datasetId:002": "urn:ngsi-ld:Person:14142"
  }
}
```

NOTE: When the simplified GeoJSON representation is selected, the layout of the Entities changes, see clause 4.5.17 for details.

4.5.5 Multi-Attribute Support

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

To be able to explicitly manage such multi-attributes, the optional *datasetId* property is used, which is of datatype URI, or equal to the JSON-LD keyword *"@none"*. It is introduced for Properties and Relationships in clauses 4.5.2 and 4.5.3 respectively. If a *datasetId* is provided when creating, updating, appending or deleting Attributes, only instances with the same *datasetId* are affected, leaving instances with another *datasetId* or an instance without a *datasetId* untouched. If no *datasetId* is provided, or *datasetId: "@none"* is supplied, it is considered as the default Attribute instance. Thus, the creation, updating, appending or deleting of Attributes without providing a *datasetId* only affects the default Attribute instance. There can only be one default Attribute instance for an Attribute with a given Attribute Name in any request or response. An example can be found in annex C, clause C.2.2.

When requesting Entity information, if there are multiple instances of matching Attributes these are returned as arrays of Attributes, instead of a single Attribute element. The *datasetId* of the default Attribute instance is never explicitly included in responses.

There is no multi-attribute support for non-reified Attributes, in particular this applies to the Temporal Properties *createdAt*, *modifiedAt*, *deletedAt* and *observedAt*, and also the *unitCode* Property.

In case of conflicting information for an Attribute, where a *datasetId* is duplicated, but there are differences in the other attribute data, the one with the most recent *observedAt* DateTime, if present, and otherwise the one with the most recent *modifiedAt* DateTime shall be provided. If no other mechanism for determining the most current Attribute instance is found, the NGSI-LD system shall choose the Attribute instance at random and the result is indeterminate.

4.5.6 Temporal Representation of an Entity

The temporal representation of an Entity shall be as mandated by clause 4.5.1, but for each Property and Relationship their temporal representation shall be provided as mandated by clauses 4.5.7 and 4.5.8 and the Scope (if present) shall be represented as a Temporal Representation of a Property (clause 4.5.7) that can only have the non-reified Temporal Properties *createdAt, modifiedAt, deletedAt* and *observedAt* as sub-Properties. This is required to represent the temporal evolution of the Scope. In case the Temporal Representation of the Scope is updated as the result of a change from the Core API, the *observedAt* sub-Property should be set as a copy of the *modifiedAt* sub-Property.

4.5.7 Temporal Representation of a Property

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

The temporal evolution of an NGSI-LD Property shall be represented as an Array of JSON-LD objects, each one representing an instance of the Property (as mandated by clause 4.5.2) at a particular point in time, which is recorded as a Temporal Property of the instance (typically "observedAt"). See example in annex C, clause C.5.6. In case the Property is deleted, an instance of the Property is recorded with its *value* set to the URI *"urn:ngsi-ld:null"* and the *deletedAt* Temporal Property set.

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

4.5.8 Temporal Representation of a Relationship

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

The temporal evolution of an NGSI-LD Relationship shall be represented as an Array of JSON-LD objects, each one representing an instance of the Relationship (as mandated by clause 4.5.3) at a particular point in time, which is recorded as a Temporal Property of the instance (typically "observedAt"). See example in annex C, clause C.5.5. In case the Relationship is deleted, an instance of the Relationship is recorded with its *object* set to the URI *"urn:ngsi-ld:null"* and the *deletedAt* Temporal Property set.

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

4.5.9 Simplified Temporal Representation of an Entity

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

The simplified temporal representation of an Entity shall be a JSON-LD object containing the following members:

- "id" whose value shall be a URI that identifies the Entity. *Mandatory*.
- "type" whose value shall be equal to the Entity Type Name or an unordered JSON array with multiple Entity Type Names in case of an Entity that has multiple Entity Types. *Mandatory*.
- "@context", a JSON-LD @context as described in clause 4.4. *Optional*.

- For each **Property**, a member whose key is the Property Name (a term). The member value shall be a JSON-LD object labelled with the type "Property". Such JSON-LD object shall only contain a member whose key shall be "values". The value of the referred values member shall be a JSON-LD Array that shall contain as many array elements as Property instances (i.e. data points of the concerned Property) being represented. Each array element shall be another Array containing exactly two array elements: the first element shall be a Property value and the second element shall correspond to the associated Temporal Property (for instance "observedAt").

EXAMPLE 1:

```
"name": {
  "type": "Property",
  "values": [
    [
      "Joe Bloggs",
      "2022-08-09T18:25:02Z"
    ],
    [
      "Bill Smith",
      "2022-08-10T18:25:02Z"
    ]
  ]
}
```

- For each **GeoProperty**, a member whose key is the Property Name (a term). The member value shall be a JSON-LD object labelled with the type "GeoProperty". Such JSON-LD object shall only contain a member whose key shall be "values". The value of the referred values member shall be a JSON-LD Array that shall contain as many array elements as GeoProperty instances (i.e. data points of the concerned GeoProperty) being represented. Each array element shall be another Array containing exactly two array elements: the first element shall be a GeoProperty value and the second element shall correspond to the associated Temporal Property (for instance "observedAt").

- For each **LanguageProperty**, a member whose key is the Property Name (a term). The member value shall be a JSON-LD object labelled with the type "LanguageProperty". Such JSON-LD object shall only contain a member whose key shall be "languageMaps". The value of the referred languageMaps member shall be a JSON-LD Array that shall contain as many array elements as LanguageProperty instances (i.e. data points of the concerned LanguageProperty) being represented. Each array element shall be another Array containing exactly two array elements: the first element shall be a JSON Object containing a single Attribute with a key called "languageMap" where the value shall correspond to a LanguageProperty languageMap and the second element shall correspond to the associated Temporal Property (for instance "observedAt").

EXAMPLE 2:

```
"says": {
  "type": "LanguageProperty",
  "languageMaps": [
    [
      {"languageMap": {"en": "yes", "fr": "oui"}},
      "2022-08-09T18:25:02Z"
    ],
    [
      {"languageMap": {"en": "no", "fr": "non"}},
      "2022-08-10T18:25:02Z"
    ]
  ]
}
```

- For each **VocabularyProperty**, a member whose key is the Property Name (a term). The member value shall be a JSON-LD object labelled with the type "VocabularyProperty". Such JSON-LD object shall only contain a member whose key shall be "vocabs". The value of the referred *vocabs* member shall be a JSON-LD Array that shall contain as many array elements as VocabularyProperty instances (i.e. data points of the concerned VocabularyProperty) being represented. Each array element shall be another Array containing exactly two array elements: the first element shall be a JSON Object containing a single Attribute with a key called "vocab", where the value shall correspond to a VocabularyProperty vocab and the second element shall correspond to the associated Temporal Property (for instance "observedAt").

EXAMPLE 3:

```
"gender": {
  "type": "VocabularyProperty",
  "vocabs": [
    [
      {"vocab": "Male"},
      "2022-08-09T18:25:02Z"
    ],
    [
      {"vocab": "Female"},
      "2022-08-10T18:25:02Z"
    ]
  ]
}
```

- For each **Relationship**, a term whose key is the Relationship Name (a term). The member value shall be a JSON-LD object labelled with the type "Relationship". Such JSON-LD object shall only contain a member whose key shall be "objects". The value of the referred *objects* member shall be a JSON-LD Array that shall contain as many array elements as Relationship instances (i.e. data points of the concerned Relationship) being represented. Each array element shall be another array containing exactly two elements: the first element shall be a Relationship object (a URI) and the second element shall correspond to the associated Temporal Property (for instance "observedAt").

EXAMPLE 4:

```
"spouse": {
  "type": "Relationship",
  "objects": [
    [
      "urn:ngsi-ld:Person:123455",
      "2022-08-09T18:25:02Z"
    ],
    [
      "urn:ngsi-ld:Person:999999",
      "2022-08-10T18:25:02Z"
    ]
  ]
}
```

4.5.10 Entity Type List Representation

The entity type list representation is used to consume information about entity types. The entity type list representation shall be a JSON-LD object containing the following members:

- "id" whose value shall be a URI that identifies the entity type list. *Mandatory*.
- "type": "EntityTypeList". *Mandatory*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- "typeList": JSON-LD array containing the entity type names. *Mandatory*.

4.5.11 Detailed Entity Type List Representation

The detailed entity type list representation is used to consume detailed information about entity types including the names of attributes that instances of each entity type can have. The detailed entity type list representation shall be an array of JSON-LD objects containing the following members:

- "id" whose value shall be the URI that identifies the entity type. *Mandatory*.
- "type": "EntityType". *Mandatory*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- "typeName": Name of entity type, short name if contained in @context. *Mandatory*.

- "attributeNames": JSON-LD array containing the names of attributes that instances of the entity type can have. *Mandatory*.

4.5.12 Entity Type Information Representation

The entity type information representation is used to consume detailed information about an entity type. The entity type information representation shall be a JSON-LD object containing the following members:

- "id" whose value shall be the URI that identifies the entity type. *Mandatory*.
- "type": "EntityTypeInfo". *Mandatory*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- "typeName": the URI that identifies the entity type (short name in case of availability in @context). *Mandatory*.

4.5.13 Attribute List Representation

The attribute list representation is used to consume information about attributes. The attribute list representation shall be a JSON-LD object containing the following members:

- "id" whose value shall be a URI that identifies the attribute list. *Mandatory*.
- "type": "AttributeList". *Mandatory*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- "attributeList": JSON-LD array containing the attribute names. *Mandatory*.

4.5.14 Detailed Attribute List Representation

The detailed attribute list representation is used to consume detailed information about attributes including the names of entity types that have instances with attributes, which have the respective attribute name. The detailed attribute list representation shall be an array of JSON-LD objects containing the following members:

- "id" whose value shall be the URI that identifies the attribute. *Mandatory*.
- "type": "Attribute". *Mandatory*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- "attributeName": the URI that identifies the attribute (short name in case of availability in @context). *Mandatory*.
- "typeNames": an array of the names of entity types that have instances with attributes, which have the respective attribute name. *Optional*.

4.5.15 Attribute Information Representation

The attribute information representation is used to consume detailed information about an attribute. The attribute information representation shall be a JSON-LD object containing the following members:

- "id" whose value shall be the URI that identifies the attribute. *Mandatory*.
- "type": "Attribute". *Mandatory*.
- "@context" a JSON-LD @context as described in clause 4.4. *Optional*.
- "attributeName": the URI that identifies the attribute (short name in case of availability in @context). *Mandatory*.

- "attributeCount": number of instances of this attribute. *Optional*.
- "attributeTypes": an array of attribute types (e.g. Property, Relationship, GeoProperty) for which instances with the attribute name exist. *Optional*.
- "typeNames": an array of the names of entity types that have instances with attributes, which have the respective attribute name. *Optional*.

4.5.16 GeoJSON Representation of Entities

4.5.16.0 Foreword

The NGSI-LD specification defines an alternative representation of Entities, to make NGSI-LD responses compatible with GIS (Geographic Information System) applications which support the GeoJSON format [8] and/or GeoJSON-LD [i.20].

Every NGSI-LD Entity can be represented as a GeoJSON *Feature* object, where a *Feature* object represents any spatially bounded thing as defined by its geometry.

4.5.16.1 Top-level "geometry" field selection algorithm

A parameter of the request (named "geometryProperty") may be used to indicate the name of the **GeoProperty** to be selected. If this parameter is not present, then the default name of "location" shall be used.

If the selected **GeoProperty** has multiple instances as described in clause 4.5.5, either a "datasetId" shall be specified, in order to define which instance of the value is to be selected, or a default attribute instance exists, which is then selected, if no "datasetId" was specified.

If an entity lacks the **GeoProperty** as specified or the value does not hold a valid GeoJSON *geometry* object then the *geometry* shall be undefined and returned with a value of null - which is syntactically valid GeoJSON.

4.5.16.2 GeoJSON Representation of an individual Entity

The GeoJSON representation of a spatially bounded Entity is defined as a single GeoJSON *Feature* object including the following members:

- "id": *Mandatory* - the Entity "id".
- "type": *Mandatory* - the fixed value "Feature".
- "geometry": *Mandatory* - The value of the selected **GeoProperty** (a GeoJSON *geometry* object) used to define the spatial location of the Entity. Note that no sub-Attributes of the selected **GeoProperty** are present in the representation.
- "properties": *Mandatory* - A JSON object containing the following members:
  - "type": *Mandatory* - the Entity Type Name of the Entity or an unordered JSON array with the Entity Type Names of the Entity.
  - One member for each **Property** (including the selected **GeoProperty**) as per the rules stated in clause 4.5.2. In case of multiple Property instances with the same Property Name as described in clause 4.5.5, all instances are provided as an unordered JSON array.
  - One member for each **Relationship** as per the rules stated in clause 4.5.3. In case of multiple Relationship instances with the same Property Name as described in clause 4.5.5, all instances are provided as an unordered JSON array.
- A JSON-LD @context as described in clause 4.4 if requested as part of the payload body.

This representation shall be fully compliant with *Feature* as defined within IETF RFC 7946 [8].

An example can be found in annex C, clause C.2.3.

4.5.16.3 GeoJSON Representation of Multiple Entities

The GeoJSON representation of a list of spatially bounded Entities is defined as a single GeoJSON **FeatureCollection** object containing an array of GeoJSON *Feature* objects as follows:

- "type": *Mandatory* - the fixed value "FeatureCollection".
- "features": a JSON array of GeoJSON *Feature* objects as defined in clause 4.5.16.2. Note that separate @context elements for each *Feature* will not be present in the payload body.
- A JSON-LD @context as described in clause 4.4 if requested as part of the payload body.

This representation shall be fully compliant with *FeatureCollection* as defined within IETF RFC 7946 [8].

An example can be found in annex C, clause C.2.3.

4.5.17 Simplified GeoJSON Representation of Entities

4.5.17.0 Foreword

When both simplified (see clause 4.5.4) and GeoJSON representation is requested, the following simplified GeoJSON representation compatible with GIS systems shall be returned.

4.5.17.1 Simplified GeoJSON Representation of an individual Entity

The simplified GeoJSON representation of a spatially bounded Entity is defined as a single GeoJSON *Feature* object as follows:

- "id": *Mandatory* - the Entity "id".
- "type": *Mandatory* - the fixed value "Feature".
- "geometry": *Mandatory* - The value of the selected **GeoProperty** (a GeoJSON *geometry* object) used to define the spatial location of the Entity.
- "properties": *Mandatory* - An array containing the following attributes:
    - "type": *Mandatory* - the Entity Type Name of the Entity or an unordered JSON array with the Entity Type Names of the Entity.
    - For each **Property** (including the selected **GeoProperty**) a member whose key is the Property Name (a term) and whose value is the Property Value. In the multi-attribute case, each Property consists of a key-value pair, the key being the Property Name (a term) and the value being a JSON Object, which contains a single Attribute with a key called "dataset", and its value in turn is a JSON Object holding a series of key-value pairs, one for each *datasetId*, where the value corresponds to the simplified representation of the property value. The default *datasetId* (where present) is represented by the JSON-LD keyword *@none*.
    - 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). In the multi-attribute case, each Relationship consists of a key-value pair, the key being the Relationship Name (a term) and the value being a JSON Object containing a single Attribute with a key called "dataset" and its value in turn is a JSON Object holding a series of key-value pairs, one for each *datasetId* where the value corresponds to the object of the relationship. The default *datasetId* (where present) is represented by the JSON-LD keyword *@none*.
- A JSON-LD @context as described in clause 4.4 if requested as part of the payload body.

The selection of the geometry field is defined in clause 4.5.16.1.

This representation shall be fully compliant with *Feature* as defined within IETF RFC 7946 [8].

An example can be found in annex C, clause C.2.3.

4.5.17.2 Simplified GeoJSON Representation of multiple Entities

The simplified GeoJSON representation of a list of spatially bounded Entities is defined as a single GeoJSON *FeatureCollection* object containing an array of GeoJSON *Feature* objects as follows:

- "type": *Mandatory* - the fixed value "FeatureCollection".
- "features": *Mandatory* - a JSON array of simplified GeoJSON *Feature* objects as defined in clause 4.5.17.1. Note that separate @context elements for each *Feature* will not be present in the payload body.
- A JSON-LD @context as described in clause 4.4 if requested as part of the payload body.

This representation shall be fully compliant with *FeatureCollection* as defined within IETF RFC 7946 [8].

4.5.18 NGSI-LD LanguageProperty Representations

4.5.18.1 Introduction

NGSI-LD defines a specialized type of Property named *LanguageProperty*, defined by the NGSI-LD @context described by the present document in clause 4.4.

When dealing with NGSI-LD Entities, implementations shall interpret the JSON-LD nodes of type *LanguageProperty* as per clause 4.5.18.2 (when in normalized representation) or clause 4.5.18.3 (when in concise representation).

Both normalized and concise representation of LanguageProperties shall be supported by implementations and can be selected by Context Consumers through specific request parameters. An example of this representation can be found in annex C, clause C.2.2.

4.5.18.2 Normalized NGSI-LD LanguageProperty

An NGSI-LD LanguageProperty shall be represented in normalized representation by a member whose key is the LanguageProperty Name (a term), whose value is the same as the JSON-LD object in **NGSI-LD Property Representation** defined in clause 4.5.2.2, with the following differences:

- "type": "LanguageProperty". *Mandatory*.
- "languageMap": a JSON object consisting of a set of a non-empty language tags as defined by IETF RFC 5646 [28] or the language tag *"@none"* which represents a default language, with each language tag mapping to a single string or array of strings. It represents a more specialized "value". *Mandatory*.
- An NGSI-LD Null used during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) shall be encoded as the JSON object *{"@none": "urn:ngsi-ld:null"}*. The same representation is also used to indicate a deletion in notifications and in the temporal evolution for encoding a deleted LanguageProperty.
- "previousLanguageMap": only provided in case of notifications and if the *showChanges* option is explicitly requested. It represents the previous Language Property *languageMap*, before the triggering change. *Optional*. The representation is the same as that of "languageMap".
- "unitCode": shall never be present, as languageMaps are always strings and hence unitless. *Prohibited*.
- "value" and "previousValue": shall never be present, as value is a generalization of languageMap. *Prohibited.*

4.5.18.3 Concise NGSI-LD LanguageProperty

An NGSI-LD LanguageProperty shall be represented in concise but lossless representation by a member whose key is the LanguageProperty Name (a term), whose value is the same as the JSON-LD object in **NGSI-LD Property Representation** defined in clause 4.5.2.3, with the following differences:

- "type": *Optional*. If missing, "LanguageProperty" can be inferred by the presence of the "languageMap" attribute.

- "languageMap": a JSON object consisting of a set of a non-empty language tags as defined by IETF RFC 5646 [28] or the language tag *"@none"* which represents a default language, with each language tag mapping to a single string or array of strings. It represents a more specialized "value". *Mandatory*.
- An NGSI-LD Null used during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) shall be encoded as the JSON object *{"@none": "urn:ngsi-ld:null"}*. The same representation is also used to indicate a deletion in notifications and the temporal evolution for encoding a deleted LanguageProperty.
- "unitCode": shall never be present, as languageMaps are always strings and hence unitless. *Prohibited.*
- "value": shall never be present, as it is a generalization of languageMap. *Prohibited.*

Notwithstanding the definition above, during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12), an NGSI-LD LanguageProperty with a value of NGSI-LD Null without a datasetId should be represented in a concise representation by a member whose key is the LanguageProperty name (a term) and whose value is *"urn:ngsi-ld:null"*.

4.5.19 Aggregated Temporal Representation of an Entity

4.5.19.0 Foreword

The NGSI-LD specification defines an alternative temporal representation of Entities, called Aggregated Temporal Representation, which allows consuming temporal Entity data after applying an aggregation function on the values of the Attribute instances. The aggregated temporal representation of Entities shall be supported by implementations supporting the Temporal Representation of Entities and can be selected by Context Consumers through specific request parameters. An example can be found in annex C, clause C.5.14.

The aggregation function is applied according to the following principles:

- An aggregation method specifies the function used to aggregate the values (e.g. sum, mean, etc.). A Context Consumer can ask for many aggregation methods in one request.
- The duration of an aggregation period specifies the duration of each period to be used when applying the aggregation function on the values of a Temporal Entity.

The Aggregated Temporal Representation of an Entity shall include the following:

- A JSON-LD object containing the following members:
  - id, type and @context as described in clause 4.5.1.
  - For each **Property** a member whose key is the Property Name (a term). The member value shall be a JSON-LD object labelled with the type "Property". Such JSON-LD object shall contain one member per aggregation method requested by the Context Consumer. Each member uses the aggregation method name as a key. The value of each member shall be a JSON-LD Array that shall contain as many array elements as there are periods in the time range of the query. Each array element shall be another Array containing exactly three array elements in the following order:
    1) the value obtained after applying the aggregation method over the period;
    2) the start DateTime of the corresponding period;
    3) the end DateTime of the corresponding period.
  - For each **Relationship** a term whose key is the Relationship Name (a term). The member value shall be a JSON-LD object labelled with the type "Relationship". Such JSON-LD object shall contain one member per aggregation method requested by the Context Consumer. Each member uses the aggregation method name as a key. The value of each member shall be a JSON-LD Array that shall contain as many array elements as there are periods in the time range of the query. Each array element shall be another array containing exactly three array elements in the following order:
    1) the value obtained after applying the aggregation method over the period;
    2) the start DateTime of the corresponding period;

3) the end DateTime of the corresponding period.

An example of this Aggregated Temporal Representation can be found in annex C, clause C.5.14.

4.5.19.1 Supported behaviours for aggregation functions

In order to support such aggregation functions, two parameters are defined:

- aggrMethods, to express the aggregation methods to apply.
- aggrPeriodDuration to express the duration of the period to consider in each step of the aggregation.

The duration is expressed using the ISO 8601 [17] Duration Representation and in particular using the following format and conventions:

- The duration shall be a string in the format P[n]Y[n]M[n]DT[n]H[n]M[n]S or P[n]W, where [n] is replaced by the value for each of the date and time elements that follow the [n], P is the duration designator and T is the time designator. For example, "P3Y6M4DT12H30M5S" represents a duration of "three years, six months, four days, twelve hours, thirty minutes, and five seconds".
- Date and time elements including their designator may be omitted if their value is zero.
- Lower-order elements may be omitted for reduced precision.
- A duration of 0 second (e.g. expressed as "PT0S" or "P0D") is valid and is interpreted as a duration spanning the whole time range specified by the temporal query.
- Alternative representations based on combined date and time representations are not allowed.

The values supported by the **aggrMethods** parameter are the following:

- `aggrMethods = "totalCount" / "distinctCount" / "sum" / "avg" / "min" / "max" / "stddev" / "sumsq"`

The semantics of the different aggregation methods defined above is as follows, and shall be supported by compliant implementations.

Table 4.5.19.1-1: Semantics of aggregation methods for Properties on JSON native data types

Aggregation Method	JSON String	JSON Number	JSON Object	JSON Array	JSON Boolean (for the purpose of the aggregation, true is considered as a value of 1, false is considered as a value of 0)
totalCount	Calculate the number of times the value has been updated inside the period				
distinctCount	Calculate the count of distinct values inside the period				
sum	N/A	Calculate the sum of the values inside the period	N/A	Calculate the sum of the sizes of the arrays inside the period	Calculate the sum of the values inside the period
avg	N/A	Calculate the average of the values inside the period	N/A	Calculate the average number of the sizes of the arrays inside the period	Calculate the average of the values inside the period
min	Calculate the first value in lexicographical order inside the period	Calculate the minimum value inside the period	N/A	Calculate the minimum size of the arrays inside the period	Calculate the minimum value inside the period
max	Calculate the last value in lexicographical order inside the period	Calculate the maximum value inside the period	N/A	Calculate the maximum size of the arrays inside the period	Calculate the maximum value inside the period
stddev	N/A	Calculate the standard deviation of the values inside the period	N/A	N/A	Calculate the standard deviation of the values inside the period
sumsq	N/A	Calculate the sum of the square of the values inside the period	N/A	N/A	Calculate the sum of the square of the values inside the period

Table 4.5.19.1-2: Semantics of aggregation methods for Properties on other supported data types

Aggregation Method	DateTime	Date	Time	URI
totalCount	Calculate the number of times the value has been updated inside the period			
distinctCount	Calculate the count of distinct values inside the period			
sum	N/A	N/A	N/A	N/A
avg	N/A	N/A	Calculate the average time inside the period (e.g. to apply on an event that occurs at non fixed times, like the time a car enters a given parking)	N/A
min	Calculate the minimum value inside the period	Calculate the minimum value inside the period	Calculate the minimum value inside the period	N/A
max	Calculate the maximum value inside the period	Calculate the maximum value inside the period	Calculate the maximum value inside the period	N/A
stddev	N/A	N/A	N/A	N/A
sumsq	N/A	N/A	N/A	N/A

Table 4.5.19.1-3: Semantics of aggregation methods for Relationships

Aggregation Method	Relationship
totalCount	Calculate the number of times the relationship has been updated inside the period
distinctCount	Calculate the count of distinct relationships targets inside the period
sum	N/A
avg	N/A
min	N/A
max	N/A
stddev	N/A
sumsq	N/A

4.5.20 NGSI-LD VocabularyProperty Representations

4.5.20.1 Introduction

NGSI-LD defines a specialized type of Property named *VocabularyProperty*, defined by the NGSI-LD @context described by the present document in clause 4.4.

When dealing with NGSI-LD Entities, implementations shall interpret the JSON-LD nodes of type *VocabularyProperty* as per clause 4.5.20.2 (when in normalized representation) or clause 4.5.20.3 (when in concise representation).

Both normalized and concise representation of VocabularyProperties shall be supported by implementations and can be selected by Context Consumers through specific request parameters. An example of this representation can be found in annex C, clause C.2.2.

4.5.20.2 Normalized NGSI-LD VocabularyProperty

An NGSI-LD VocabularyProperty shall be represented in normalized representation by a member whose key is the VocabularyProperty Name (a term), whose value is the same as the JSON-LD object in **NGSI-LD Property Representation** defined in clause 4.5.2.2, with the following differences:

- "type": "VocabularyProperty". *Mandatory.*
- "vocab": a JSON object consisting of a single string or array of strings which can be type coerced into an IRI or array of IRIs. It represents a more specialized "value". *Mandatory*.
- "previousVocab": only provided in case of notifications and if the *showChanges* option is explicitly requested. It represents the previous VocabularyProperty *vocab*, before the triggering change. *Optional*. The representation is the same as that of "vocab".
- "unitCode": shall never be present, as vocabs are always strings and hence unitless. *Prohibited*.
- "value" and "previousValue": shall never be present, as value is a generalization of vocab. *Prohibited.*
- "object" and "previousObject": shall never be present, as they define a Relationship's object URI. *Prohibited*.

4.5.20.3 Concise NGSI-LD VocabularyProperty

An NGSI-LD VocabularyProperty shall be represented in concise but lossless representation by a member whose key is the VocabularyProperty Name (a term), whose value is the same as the JSON-LD object in **NGSI-LD Property Representation** defined in clause 4.5.2.3, with the following differences:

- "type": *Optional*. If missing, "VocabularyProperty" can be inferred by the presence of the "vocab" attribute.
- "vocab": a JSON object consisting of a single string or array of strings which can be type coerced into an IRI or array of IRIs. It represents a more specialized "value". *Mandatory*.
- "unitCode": shall never be present, as vocabs are always strings and hence unitless. *Prohibited.*
- "value": shall never be present, as it is a generalization of vocab. *Prohibited.*

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, NGSI-LD restricts Entity Type Names, Property Names and Relationship Names to the following ABNF grammar:

```
nameChar = unicodeNumber / unicodeLetter
nameChar =/ %x5F                              ; _
name = unicodeLetter *nameChar
```

- *unicodeNumber* is any Unicode character that has *Number* as a Category [22]. With Unicode-capable regular expression (RegEx) parsers, such a character may be matched by \p{N}.
- *unicodeLetter* is any Unicode character that has *Letter* as a Category [22]. With Unicode-capable regular expression (RegEx) parsers, such a character may be matched by \p{L}.

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

```
prefix = unicodeLetter *nameChar
name =/ prefix %x3A unicodeLetter *nameChar ; prefix:name
```

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

4.6.3 Supported data types for Values

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

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

NOTE 1: In previous versions of NGSI-LD, only the comma was supported as ISO 8601 [17] states that it is the preferred option. However, in practice the decimal point is more commonly used.

  - The trailing timestamp component shall contain the time zone related information and shall always be equal to the character "Z". Therefore, all timestamps shall be expressed in **UTC**.

- **Date** string for encoding a calendar date. It uses ISO 8601 [17] Complete Representation using the 'Extended Format', as described below:
  - It shall be a string containing *Year*, *Month*, *Day* components using the format *YYYY-MM-DD* as defined in ISO 8601 [17]. In this representation, the character "-" is used to separate the calendar date components.
  - All the referred components shall appear in the string; reduced representations are not permitted.
- **Time** string for encoding a local time expressed in **UTC**. It uses ISO 8601 [17] Complete Representation using the 'Extended Format', as described below:
  - It shall be a string containing *Hours*, *Minutes* and *Seconds* components using the format *hh:mm:ssZ* as defined in ISO 8601 [17]. In this representation, the character ":" is used to separate the local time components.
  - All the referred components shall appear in the string; reduced representations are not permitted.
  - The *Seconds* component may optionally contain a decimal fraction. In this case the string shall contain two integer digits, followed by a decimal point and then one or more fractional digits, up to a maximum of six. For example, *hh:mm:ss.ssssssZ.* In requests, also a comma instead of a decimal point may be used as separator for compatibility reasons.

NOTE 2: In previous versions of NGSI-LD, only the comma was supported as ISO 8601 [17] states that it is the preferred option. However, in practice the decimal point is more commonly used.

  - 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 (e.g. a set, a list).

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*.

4.6.5 Supported data types for LanguageMaps

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

- A JSON object consisting of a series of key-value pairs where the keys shall be JSON strings representing IETF RFC 5646 [28] language codes or the JSON-LD *"@none"* for representing default when no more specific language is found. and the values shall be JSON strings or arrays of JSON strings. Additionally the languageMap encoding *{"@none": "urn:ngsi-ld:null"}* shall be used to represent an NGSI-LD Null during partial update patch and merge patch (see clauses 5.5.8 and 5.5.12) and for representing deleted Language Properties in notifications and temporal evolutions.

4.6.6 Ordering of Entities in arrays having more than one instance of the same Entity

Some services (batch operations, clauses 5.6.7, 5.6.8, 5.6.9 and 5.6.10) operate on an array of entities, as input, and if this array contains more than one instance of the same entity, then these entity instances shall come in chronological order, i.e. the first entity instance in the array shall be older than the second, the second shall be older than the third, etc.

Without this assumption, there is no way for the request to be treated correctly, as the entity instances are often used for replacing or modifying the prior entity instance.

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 Core NGSI-LD @context described by the present document in clause 4.4.

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

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

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

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

4.7.2 Representation of GeoJSON Geometries in JSON-LD

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

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

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

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

4.7.3 Concise NGSI-LD GeoProperty

Notwithstanding the restrictions defined in clause 4.5.2.3, an NGSI-LD GeoProperty without additional sub-attributes shall be represented in a concise but lossless representation by a member whose key is the Property Name (a term) and whose value is the Property Value (see definition of terms in clause 3.1) which itself is also a supported GeoJSON geometry.

- "type": shall be a supported GeoJSON geometry type as defined in clause 4.7.1. *Mandatory*.
- "coordinates": shall be present, as defined by the relevant GeoJSON Geometry [8]. *Mandatory*.

When parsing a geospatial value submitted in the concise representation, it shall be possible for the NGSI-LD system to infer the GeoProperty type. Error handing of the payload is left ambiguous if the NGSI-LD system is unable to distinguish a payload as either a Property or a GeoProperty.

Furthermore, an NGSI-LD GeoProperty which includes additional Properties or Relationships shall be treated in the same manner as an ordinary NGSI-LD Property (see clause 4.5.2.3) with the exception that if the Property Value resolves to a supported GeoJSON geometry, the type "GeoProperty" shall be inferred.

4.8 Temporal Properties

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

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

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

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

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

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

Whenever a *TemporalProperty* value is unknown by a registered Context Source, the Property shall be omitted rather than sending an error response.

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 is the "value" member of a Property, see table 5.2.5-1, or the "object" member of a Relationship, see table 5.2.6-1);
- filter out Context Sources by the values of properties that describe them, defined when Context Sources are registered (target is the name of a Context Source Property member of the CSourceRegistration, see table 5.2.9-1).

In this clause, two string parameters are defined in order to fully specify an NGSI-LD Query:

- **q**, to express the desired query;
- **expandValues**, to define the list of attributes whose values should be expanded against the supplied @context using JSON-LD type coercion prior to executing the query in the broker. *Optional.*

In case of HTTP binding, whenever the string acting as a filter is part of the HTTP binding's URI, then it shall be URI-encoded (percent-encoded, as described in IETF RFC 3986 [5]).

The grammar that encodes the syntax of the **q** parameter, expressed in ABNF format [12], is the NGSI-LD Query Language. It is described below (it has been validated using https://tools.ietf.org/tools/bap/abnf.cgi), and it shall be supported by implementations:

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

- *unicodeNumber* is any Unicode character that has *Number* as a Category [22]. With Unicode-capable regular expression (RegEx) parsers, such a character may be matched by \p{N}.
- *unicodeLetter* is any Unicode character that has *Letter* as a Category [22]. With Unicode-capable regular expression (RegEx) parsers, such a character may be matched by \p{L}.
- *Number* shall be a number as mandated by the JSON Specification, following the ABNF Grammar, production rule named *number*, section 6 of IETF RFC 8259 [6].
- *String* shall be a text string as mandated by the JSON Specification, following the ABNF Grammar, production rule named *String*, section 7 of IETF RFC 8259 [6].

- *char* shall be a character as mandated by the JSON Specification, ABNF Grammar, production rule named *char*, section 7 of IETF RFC 8259 [6].
- *false* shall be conformant with the JSON ABNF Grammar, production rule named *false*, section 3 of IETF RFC 8259 [6]. It is intended to represent the Boolean value corresponding to "false".
- *true* shall be conformant with the JSON ABNF Grammar, production rule named *true*, section 3 of IETF RFC 8259 [6]. It is intended to represent the Boolean value corresponding to "true".
- *RegExp* shall be a regular expression as mandated by IEEE 1003.2™ [11].
- *dateTime* shall be a *DateTime* value as mandated by clause 4.6.3.
- *time* shall be a *Time* value as mandated by clause 4.6.3.
- *date* shall be a *Date* value as mandated by clause 4.6.3.
- *URI* shall be a URI as mandated by IETF RFC 3986 [5] or an IRI as mandated by IETF RFC 3987 [23], appendix A, production rule named *URI*.

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

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

The attribute path (production rule *Attribute*) is a simple name *AttrName*, optionally followed by a dot-separated list of more *AttrName* (see later Example 8), optionally followed by one trailing list of more dot-separated *AttrName*s enclosed in one pair of square brackets (see later Example 9). The attribute path is always a composition of short hand names and not a fully qualified ones, because, when the query language is used, an @context properly defining all the terms (as per clause 5.5.7) shall be issued.

EXAMPLE 0: ?**q**=temperature. (checks for the existence of the attribute temperature).

EXAMPLE 1: ?**q**=temperature==20.

EXAMPLE 2: ?**q**=brandName!="Mercedes".

EXAMPLE 3: ?**q**=isParked=="urn:ngsi-ld:OffStreetParking:Downtown1".

EXAMPLE 4: A query encoded as an HTTP Query String. Note that this is HTTP binding specific, to be used via GET method, as defined in clause 6.4.3.2. The NGSI-LD query language string is conveyed by means of parameter **q**.

?**q**=speed>50;brandName!="Mercedes". Also note that (as stated above) URI-encoding (percent-encoding) is required if the query string contains reserved characters (see IETF RFC 3986 [5] and IETF RFC 3987 [23], for the exact list of them).

EXAMPLE 5: ?**q**=isMonitoredBy (to query Entities that have the Attribute isMonitoredBy).

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

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

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

EXAMPLE 6: ?**q**=((speed>50|rpm>3000);brandName=="Mercedes").

EXAMPLE 7: ?**q**=(temperature>=20;temperature<=25)|capacity<=10.

The following Example 8 shows the syntax of an attribute path that is defined by the production rule *Attribute*, as a dot-separated list of names. Such a list is intended to address a Property or Relationship included by the matching entities subjacent graph, in accordance with the following rules:

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

EXAMPLE 8: ?**q**=temperature.observedAt>=2017-12-24T12:00:00Z.

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

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

If the target element is a LanguageProperty, and no target language is specified, the **target value** is defined as a value from any of the key-value pairs held within the *languageMap* associated to such LanguageProperty.

If the target element is a LanguageProperty and a target language is specified, the **target value** is defined as the Value associated to the matching key-value pair held within the *languageMap* associated to such LanguageProperty where the key matches the target language.

If the target element is a VocabularyProperty, the **target value** shall be expanded according to the @context.

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

Lastly, implementations shall support queries involving specific data subitems belonging to a Property Value (**seed target value**) represented by a JSON object structure (compound value). For that purpose, an attribute path may additionally contain a **trailing path** (enclosed in a single pair of square brackets that signal that the overall path is now entering the compound value) composed of a dot-concatenated list of JSON member names, and intended to address a specific data subitem (member) within the **seed target value.** When such a trailing path is present, implementations shall interpret and evaluate it (against the seed target value) as a *MemberExpression* of ECMA 262 [21], in dot notation, as clarified therein at section Property Accessors). If the evaluation of such *MemberExpression* does not result in a defined value, the target element shall be considered as non-existent for the purpose of query resolution.

EXAMPLE 9: ?**q**=address[city]=="Berlin". The trailing path is [city]. It is used to refer to a particular subitem within the value of the "address" Property, which is a complex JSON object representing a postal address. Refer to the following NGSI-LD Entity:

```
{
  "id":"urn:ngsi-ld:placedescription:123",
  "type":"PlaceDescription",
  "address": {
    "type":"Property",
    "value": {
```

```
      "city":"Berlin",
      "street":"Ulrich Strasse"
    }
  }
}
```

EXAMPLE 10: ?**q**=sensor.rawdata[airquality.particulate]==40. The trailing path is [airquality.particulate]. The "particulate" property of the compound JSON object is targeted. Refer to the following NGSI-LD Entity:

```
{
  "id":"urn:ngsi-ld:particulatemeasurement:345",
  "type":"ParticulateMeasurement",
  "sensor": {
    "type":"Property",
    "value": 40,
    "rawdata": {
      "type":"Property",
      "value": {
        "airquality": {
          "particulate": 40,
          "PM20": 85
        }
      }
    }
  }
}
```

EXAMPLE 11: ?**q**=gender==Male&**expandValues**=gender. The trailing path is gender. The "gender" property of JSON object is targeted, but the target value is first expanded to a URI using the supplied @context. Refer to the following NGSI-LD Entity:

```
{
  "id":"urn:ngsi-ld:Person:678",
  "type":"Person",
  "gender": {
    "type":"VocabularyProperty",
    "vocab": "Male",
    }
  },
  @context": {
    "Male": "http://example.org/Male",
  }
}
```

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

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

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

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

When it comes to comparing text strings, implementations:

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

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

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

- **Existence** (only attribute is specified). A matching entity shall contain the target element.
- **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).
  - The Query Term value target element corresponds to a LanguageProperty and a natural language is specified e.g. color[en]=="red":
    - a match is found as the value of the key-value pair corresponding to the specified natural language of the languageMap (e.g. matches *{"fr": "rouge", "en" : "red","de": "rot"}* but not *{"fr": "red", "en" : "black","de": "blue"}*).
    - a match is found as a single element from the array of values of the key-value pair corresponding to the specified natural language of the languageMap (e.g. matches *{"fr": ["chat", "rouge"], "en" : ["red", "cat], "de": ["rote", "Katze"]}* but not *{"fr": ["chat", "rouge"], "en" : ["coal", "black"],"de": ["blaue", "Engel"]}*).
  - The Query Term value target element corresponds to a LanguageProperty and no natural language is specified e.g. color[*]=="red":
    - any match is found in the values of the key-value pairs of the languageMap (e.g. matches *{"fr": "rouge", "en" : "red", "de": "rot"}.*
    - a match is found as a single element of the array of values of the key-value pairs of the languageMap (e.g. matches *{"fr": "chat", "rouge"], "en" : ["red", "cat"], "de": ["rote", "Katze"]}*).
  - The Query Term value is a URI and the target element corresponds to a VocabularyProperty, e.g. color == "http://example/red":
    - Is identical to the target value (e.g. matches *"http://example.com/red"*).
    - Is included in the target value, if the latter is an array (e.g. matches *["http://example.com/blue"," http://example.com/red"," http://example.com/green"]*).

- If the Query Term value target element corresponds to a VocabularyProperty and is a list of URIs (production rule named *ValueList*), e.g. color==" http://example/black", " http://example/red":
  - The target value is identical to any of the list values (e.g. matches *" http://example.com/red"*).
  - The target value includes any of the Query Term values, if the target value is an array (e.g. matches ["*http://example.com/red*", "http://example.com/*blue*"]).
- 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).
  - The Query Term value target element corresponds to a LanguageProperty and a natural language is specified e.g. color[en]!="red":
    - no matching value is found as the value of the specified language key of a languageMap where a language filter is specified. (e.g. matches *{"fr": "noir", "en" : "black","de": "schwarz"}*) but not {"fr": "rouge", "en" : "red","de": "rot"}.
    - no matching value is found as a single element from the array of values of the key-value pair corresponding to the specified natural language of the languageMap (e.g. matches *{"fr": ["chat", "rouge"], "en" : ["coal", "black"], "de": ["blaue", "Engel"]}* but not *{"fr": ["rouge", "noir"], "en" : ["red", "black"],"de": ["rot", "schwarz"]}*).
  - The Query Term value target element corresponds to a LanguageProperty and no language filter is specified e.g. color[*]!="red":
    - no matching value is found in any of the values of the key-value pairs of a languageMap (e.g. matches *{"fr": "noir", "en" : "black","de": "schwarz"}*) but not *{"fr": "rouge", "en" : "red","de": "rot"}*.)
    - no matching value is found as a single element from the array of values of the key-value pair corresponding to the specified natural language of the languageMap (e.g. matches *{"fr": ["chat", "rouge"], "en" : ["coal", "black"], "de": ["blaue", "Engel"]}* but not *{"fr": ["rouge", "noir"], "en" : ["red", "black"],"de": ["rot", "schwartz"]}*).
  - The Query Term value is a URI and the target element corresponds to a VocabularyProperty, e.g. color!= "http://example.com/red":
    - Is not identical to the target value (e.g. matches *"http:/example.com/black"*).
    - Is not included in the target value, if the latter is an array (e.g. matches *["http:/example.com/blue"," http:/example.com/black"," http:/example.com/green"],* but not *["http:/example.com/blue", http:/example.com/"red"," http:/example.com/green"]).*

- If the Query Term value target element corresponds to a VocabularyProperty and is a list of URIs e.g. color!= " http://example.com/black", " http://example.com/red":
  - The target value is not identical to any of the list values (e.g. matches *" http:/example.com/blue"*).
  - The target value does not include any of the list values, if the target value is an array (e.g. matches *["http:/example.com/blue"," http:/example.com/yellow"," http:/example.com/green"]*, but not *["http:/example.com/blue"," http:/example.com/red"," http:/example.com/green"]*).
- If the data type of the target value and the data type of the Query Term value are different, then they shall be considered unequal.

- **Greater than** operator (production rule named *greater*). For an entity to match, it shall contain the target element and the target value has to be strictly greater than the Query Term value:
  - If there is no equality between the target value data type and the Query Term value data type then it shall be considered as not matching.
- **Less than** operator (production rule named *less*). For an entity to match, it shall contain the target element and the target value shall be strictly less than the value:
  - If there is no equality between the target value data type and the Query Term value data type then it shall be considered as not matching.
- **Greater or equal than** (production rule named *greaterEq*). A matching entity shall meet any of the *Greater than* or the *Equal* conditions for single values.
- **Less or equal than** (production rule named *lessEq*). A matching entity shall meet any of the *Less than* or the *Equal* conditions for single values.
- **Match pattern** (production rule named *patternOp*). A matching entity shall contain the target element and the target value shall be in the L(R) of the regular pattern specified by the Query Term:
  - If the target value data type is different than String then it shall be considered as not matching.
- **Do not match pattern** (production rule named *notPatternOp*). A matching entity shall contain the target element and the target value shall not be in the L(R) of the regular pattern specified by the Query Term:
  - If the target value data type is different than String then it shall be considered as not matching.

4.10 NGSI-LD Geoquery Language

The NGSI-LD Geoquery 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.

A total of four parameters are defined in order to fully specify an NGSI-LD Geoquery:

- **georel**, to express the desired geospatial relationship;
- **geometry**, to express the type of the reference geometry;
- **coordinates**, to express the reference geometry;
- **geoproperty**, to express the target geometry of an Entity. This parameter is optional, *location* is the default.

The following grammar defines the syntax for the geospatial relationships (parameter name **georel**):

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

```
containsRel = "contains"
intersectsRel = "intersects"
equalsRel = "equals"
disjointRel = "disjoint"
overlapsRel = "overlaps"
```

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

Reference geometries shall be specified by:

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

Target geometry, i.e. the target Entity's *GeoProperty* to which the geoquery is to be applied, can be specified by an extra parameter named **geoproperty**. The GeoProperty's name shall be specified as short hand name and not a fully qualified one, because, when the query language is used, an @context properly defining all the terms (as per clause 5.5.7) shall be issued. If no *geoproperty* is specified, the geoquery is applied to the default Property *location* (see clause 4.7.1).

Note that proper URL encoding shall be used by HTTP binding API clients when using these examples.

EXAMPLE 1: **georel**=near;maxDistance==2000

geometry=Point

coordinates=[8,40]

geoproperty=observationSpace

EXAMPLE 2: **georel**=within

geometry=Polygon

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

geoproperty=location

EXAMPLE 3: A query encoded as an HTTP Query String. Note that this is HTTP binding specific, to be used via GET method, as defined in clause 6.4.3.2.

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

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

4.11 NGSI-LD Temporal Query Language

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

The following grammar defines the syntax that shall be supported:

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

The points in time for comparison are defined as follows:

- A **timeAt** parameter, 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 **endTimeAt** parameter, which is only used for the *between* relation and shall represent the end point for comparison. It shall be represented as *DateTime* (mandated by clause 4.6.3).

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

EXAMPLE 1: **timerel**=before

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

EXAMPLE 2: **timerel**=between

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

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

timeproperty=modifiedAt

EXAMPLE 3: Temporal query encoded as HTTP Query String, note that this is HTTP binding specific, to be used via GET method, as defined in clause 6.18.3.2.

?**timerel**=between&**timeAt**=2017-12-13T14:20:00Z&**timeproperty**=observedAt

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

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

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

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

4.12 NGSI-LD Pagination

NGSI-LD operations can potentially return a result set including a large number of NGSI-LD Elements, so that pagination of query results shall be supported by compliant implementations.

The list of operations that incur this behaviour is as follows:

- Query Entities (clause 5.7.2)
- Query Subscriptions (clause 5.8.4)
- Query Context Source Registrations (clause 5.10.2)
- Query Context Source Registration Subscriptions (clause 5.11.5)
- Query Temporal Evolution of Entities (clause 5.7.4)

Nonetheless, the NGSI-LD API is agnostic about specific pagination mechanisms and only defines the behaviour that shall be observed by NGSI-LD Systems.

For each operation above, NGSI-LD Systems shall:

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

NGSI-LD implementations should:

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

NGSI-LD implementations may:

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

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

4.13 Counting the Number of Results

Given that NGSI-LD Query operations can potentially return a result set including a large number of NGSI-LD Elements and that pagination of query results shall be supported (see clause 4.12), compliant implementations shall also support a mechanism for relaying to the client the number of expected resulting elements, when a query is executed.

A specific field (e.g. a custom header in the response in case of HTTP binding, see clause 6.3.13) shall be returned within the response of a query, whenever this is requested by the client.

Mechanisms for limiting the number of returned NGSI-LD Elements are independent of the counting mechanism, so that, potentially, a client can issue a query that limits to zero the number of desired results but asks for the count to be present.

This is useful for client-side planning and fine-tuning of subsequent queries and their parameters.

4.14 Supporting Multiple Tenants

The concept of a tenant is that a user or group of users utilizes a single instance of an NGSI-LD system (Context Source or Context Broker) in isolation from other users or groups of users of the same instance, which are considered to be different tenants. Thus a multi-tenant NGSI-LD system is a system where a single software instance is used by different users or groups of users, the tenants, where any information related to one tenant (e.g. Entities, Subscriptions, Context Source Registrations) are only visible to users of the same tenant, but not to users of a different tenant. Typically, multi-tenancy is used together with an access control mechanism, enforcing the isolation of tenants, however access control and other security-related aspects are out-of-scope of the present document.

The NGSI-LD API optionally enables multi-tenant systems. To support this, tenant information can be optionally specified in NGSI-LD API operations. The operation then only applies to the targeted tenant. As all information of one tenant is isolated from other tenants, the NGSI-LD API operations for managing, retrieving and subscribing to entity information, but also any context source related operations only apply to the information of the specified tenant in isolation and never have any effect on the information of other tenants.

As the support and use of tenants is optional, any operation not explicitly specifying a tenant targets a default tenant, which always exists. NGSI-LD systems not supporting multiple tenants should raise an error of type *NoMultiTenantSupport* if a tenant is specified. To enable Context Sources to be part of tenant-based distributed or federated systems, tenant information can optionally be specified in Context Source Registrations. When contacting the respective Context Sources, the tenant information from the Context Source Registration has to be used. If no tenant information is present in the Context Source Registration, no tenant information is to be used and thus the default tenant is targeted on the registered Context Source. This enables integrating Context Sources not supporting multi-tenancy in a distributed system with a tenant-based Context Broker or integrating local tenants in a federated system using a different tenant.

4.15 NGSI-LD Language Filter

The NGSI-LD Language Filter shall be supported by implementations. It is intended to form a mechanism which allows just one matching string value of LanguageProperties of NGSI-LD Entities to be converted to an NGSI-LD Property, where the value will be a string for the specified natural language.

The following grammar defines the syntax that shall be supported by the filter:

```
lang = langtag
```

Where the langtag is defined according to the rules as mandated by IETF RFC 5646 [28], and IETF RFC 3282 [29]. If the broker cannot serve any matching language, it shall default to any supported language. This behavior can be triggered by specifying `lang="*"` in the filter (see example 3).

In any case, the attribute in question shall be augmented with an additional non-reified subproperty `lang` indicating the actual language returned.

EXAMPLE 1: Specified natural language - return LanguageProperties as strings in English only.

```
lang="en"
```

EXAMPLE 2: Multiple natural languages with no ranked preference - return LanguageProperties as strings in either Swiss French or French.

```
lang="fr-CH,fr"
```

EXAMPLE 3: Wildcard - return LanguageProperties as a string in any supported language.

```
lang="*"
```

EXAMPLE 4: Quality value ranking - return all LanguageProperties as a string in Swiss French or French with no ranked preference, fallback to English as a second choice and finally fallback to any other supported language.

```
lang="fr-CH,fr;q=0.9,en;q=0.8,*;q=0.5"
```

4.16 Supporting Multiple Entity Types

From NGSI-LD API version 1.5.1 onwards, multiple Entity Types for any Entity are supported. An Entity is uniquely identified by its id, so whenever information is provided for an Entity with a given id, it is considered part of the same Entity, regardless of the Entity Type(s) specified. To avoid unexpected behaviour, Entity Types can be implicitly added by all operations that update or append attributes. There is no operation to remove Entity Types from an Entity. The philosophy here is to assume that an Entity always had all Entity Types, but possibly not all Entity Types have previously been known in the system. The only option to remove an Entity Type is to delete the Entity and re-create it with the same id. Alternatively, a batch upsert with 'replace' update mode can be used, as described in clause 5.6.8.

4.17 NGSI-LD Entity Type Selection Language

The NGSI-LD Entity Type Selection Language shall be supported by implementations. It is intended to select only those Entities that have the specified Entity Type(s), possibly among others. Entity Types are specified as a disjunction of elements, where each element can either directly be an Entity Type or a conjunction of multiple Entity Types. The logical operators are the same as in the NGSI-LD Query Language specified in clause 4.9. As a disjunction of Entity Types can also be seen as a list, and to be compatible with previous versions of the NGSI-LD API, a comma can be used as an alternative representation of the *or* operator. For logical *and* grouping parenthesis are needed.

```
EntityTypes = OrEntityType *(orOp OrEntityType)                    ; OrEntityType|OrEntityType
OrEntityType = %x28 EntityType *(andOp EntityType) %x29            ; (EntityType;EntityType)
OrEntityType = EntityType                                          ; EntityType
andOp = %x3B                                                       ; ;
orOp = %x7C / %x2C                                                 ; |  ,
```

`EntityType` is either a valid name as specified in clause 4.6.2 or a URI.

EXAMPLE 1: Entities of type Building or House:

```
Building|House
```

Alternative Representation:

```
Building,House
```

EXAMPLE 2: Entities of type Home and Vehicle:

```
(Home;Vehicle)
```

EXAMPLE 3: Entities of type (Home and Vehicle) or Motorhome:

```
(Home;Vehicle)|Motorhome
```

Alternative Representation:

```
(Home;Vehicle),Motorhome
```

NOTE: The special characters ",", ";", "(" and ")" used in the Entity Type Selection Language are allowed characters in URIs. The use of short names is recommended.

4.18 NGSI-LD Scopes

An NGSI-LD Scope enables putting Entities into a hierarchical structure and scoping queries and subscriptions according to it. The hierarchical structure is user-defined, e.g. according to (logical) location or organization. The use of Scopes is optional and an Entity can be assigned to one or more Scopes at the same time. The Scope is represented as a special *scope* Property that is non-reified in the normalized NGSI-LD Entity representation and reified in the Temporal Representation. In the latter case, it is restricted to having the non-reified Temporal Properties *createdAt, modifiedAt and deletedAt* as sub-Properties. There shall at most be one instance of the *scope* property per Entity. In case multiple representations of the same Entity have to be merged, e.g. when combining the results of distributed queries, the values of *scope* are merged. *The* value of *scope* is represented as a JSON array in case there is more than one Scope. For the Temporal Evolution a given Scope is considered valid from the time it has been set until the time it has been explicitly removed by an update or delete operation (for an example see annex C, clause C.5.16).

The grammar that encodes the syntax of the Scope is expressed in ABNF format [12]. It is described below (it has been validated using https://tools.ietf.org/tools/bap/abnf.cgi), and it shall be supported by implementations:

```
Scope = [%x2F] ScopeLevel *(%x2F ScopeLevel)                    ; [/] ScopeLevel *(/ScopeLevel)
ScopeLevel = unicodeLetter *ScopeLevelChar
ScopeLevelChar = unicodeNumber / unicodeLetter
ScopeLevelChar =/ %x5F                                          ; _
```

EXAMPLE 1: `/Madrid`

EXAMPLE 2: ` Madrid`

EXAMPLE 3: `/Madrid/Gardens/ParqueNorte`

EXAMPLE 4: `/CompanyA/OrganizationB/UnitC`

4.19 NGSI-LD Scope Query Language

The NGSI-LD Scope Query Language shall be supported by implementations. It is intended to select only those Entities that are within the specified Scope(s). Scopes are specified as a disjunction of elements, where each element can either directly be a Scope or a conjunction of multiple Scopes. The "+" can be used as a wildcard to match a single scope level within a Scope. The "#" that can be added at the end of a Scope specification serves as a wildcard, which matches the given scope and the whole hierarchy of scopes below. The "/#" matches any non-empty scope, i.e. only explicitly specified scopes. The logical operators are the same as in the NGSI-LD Query Language specified in clause 4.9. As a disjunction of Scopes can also be seen as a list, a comma can be used as an alternative representation of the *or* operator. For logical *and* grouping parenthesis are needed.

```
ScopesQ = OrScopeQ *(orOp OrScopeQ)                             ; OrScopeQ|OrScopeQ
ScopesQ =/ %x2F %23                                             ; / #
OrScopeQ = %x28 ScopeQ *(andOp ScopeQ) %x29                     ; (ScopeQ;ScopeQ)
OrScopeQ =/ ScopeQ *1(%x2F %23)                                 ; ScopeQ / #
andOp = %x3B                                                    ; ;
orOp = %x7C / %x2C                                              ; |  ,
ScopeQ = %x2F ScopeQLevel *(%x2F ScopeQLevel)                   ; /ScopeQLevel *(/ScopeQLevel)
ScopeQLevel = unicodeLetter *ScopeQLevelChar
ScopeQLevel =/ %x2B                                             ; +
ScopeQLevelChar = unicodeNumber / unicodeLetter
ScopeQLevelChar =/ %x5F                                         ; _
```

EXAMPLE 1: Scope /Madrid:

`/Madrid`

EXAMPLE 2: Scope /Madrid/Gardens and the whole scope tree below:

`/Madrid/Gardens/#`, e.g. matches the following Scopes:

`/Madrid/Gardens, /Madrid/Gardens/ParqueNorte,`
`/Madrid/Gardens/ParqueNorte/Parterre1, /Madrid/Gardens/ParqueSur`

EXAMPLE 3: Scopes /Madrid/Gardens/ParqueNorte and /Madrid/Sights/ParqueNorte, or any other Scope with Madrid as first scope level and ParqueNorte as third scope level:

```
/Madrid/+/ParqueNorte
```

EXAMPLE 4: Scope /Madrid/Districts and /CompanyA:

```
(/Madrid/Districts;/CompanyA)
```

EXAMPLE 5: Scope (/Madrid/Districts and /CompanyA) or /CompanyB:

```
(/Madrid/Districts;/CompanyA)|CompanyB
```

Alternative Representation:

```
(/Madrid/Districts;/CompanyA),CompanyB
```

4.20 NGSI-LD Distributed Operation Names

When registering Context Sources (see clause 5.2.9), the registrant NGSI-LD interface endpoint may optionally offer a subset of NGSI-LD operations which it accepts. Table 4.20-1 defines a list of names for each of these operations.

Table 4.20-1: Names of implemented Operations

	Operation name	Implements
Context Information Provision	createEntity	5.6.1 Create Entity
	updateEntity	5.6.2 Update Attributes
	appendAttrs	5.6.3 Append Attributes
	updateAttrs	5.6.4 Partial Attribute update
	deleteAttrs	5.6.5 Delete Attribute
	deleteEntity	5.6.6 Delete Entity
	createBatch	5.6.7 Batch Entity Creation
	upsertBatch	5.6.8 Batch Entity Creation or Update (Upsert)
	updateBatch	5.6.9 Batch Entity Update
	deleteBatch	5.6.10 Batch Entity Delete
	upsertTemporal	5.6.11 Create or Update Temporal Representation of an Entity
	appendAttrsTemporal	5.6.12 Add Attributes to Temporal Representation of an Entity
	deleteAttrsTemporal	5.6.13 Delete Attributes from Temporal Representation of an Entity
	updateAttrInstanceTemporal	5.6.14 Partial Update Attribute instance in Temporal Representation of an Entity
	deleteAttrInstanceTemporal	5.6.15 Delete Attribute Instance from Temporal Representation of an Entity
	deleteTemporal	5.6.16 Delete Temporal Representation of an Entity
	mergeEntity	5.6.17 Merge Entity
	replaceEntity	5.6.18 Replace Entity
	replaceAttrs	5.6.19 Attribute Replace
	mergeBatch	5.6.20 Batch Entity Merge
Context Information Consumption	retrieveEntity	5.7.1 Retrieve Entity
	queryEntity	5.7.2 Query Entities (excluding batch entity queries)
	queryBatch	5.7.2 Query Entities (batch entity queries only)
	retrieveTemporal	5.7.3 Retrieve Temporal Evolution of an Entity
	queryTemporal	5.7.4 Query Temporal Evolution of Entities
	retrieveEntityTypes	5.7.5 Retrieve Available Entity Types
	retrieveEntityTypeDetails	5.7.6 Retrieve Details of Available Entity Types
	retrieveEntityTypeInfo	5.7.7 Retrieve Available Entity Type Information
	retrieveAttrTypes	5.7.8 Retrieve Available Attributes
	retrieveAttrTypeDetails	5.7.9 Retrieve Details of Available Attributes
	retrieveAttrTypeInfo	5.7.10 Retrieve Available Attribute Information
Context Information Subscription	createSubscription	5.8.1 Create Subscription
	updateSubscription	5.8.2 Update Subscription
	retrieveSubscription	5.8.3 Retrieve Subscription
	querySubscription	5.8.4 Query Subscription
	deleteSubscription	5.8.5 Delete Subscription

In addition to these individual operations, a series of names for common groups of operations have also been defined. Table 4.20-2 defines a list of names for each of these operation groups.

Table 4.20-2: Named Operation Groups

Operation Group name	Implements
federationOps	• retrieveEntity • queryEntity • queryBatch • retrieveEntityTypes • retrieveEntityTypeDetails • retrieveEntityTypeInfo • retrieveAttrTypes • retrieveAttrTypeDetails • retrieveAttrTypeInfo • createSubscription • updateSubscription • retrieveSubscription • querySubscription • deleteSubscription
updateOps	• updateEntity • updateAttrs • replaceEntity • replaceAttrs
retrieveOps	• retrieveEntity • queryEntity
redirectionOps	• createEntity • updateEntity • appendAttrs • updateAttrs • deleteAttrs • deleteEntity • mergeEntity • replaceEntity • replaceAttrs • retrieveEntity • queryEntity • retrieveEntityTypes • retrieveEntityTypeDetails • retrieveEntityTypeInfo • retrieveAttrTypes • retrieveAttrTypeDetails • retrieveAttrTypeInfo

If no specific subset of operations is defined for a Context Source Registration, the default set of operations matches the group defined as "federationOps".

5 API Operation Definition

5.1 Introduction

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

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

5.2 Data Types

5.2.1 Introduction

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

None of the members described admit a *null* value directly, as when a JSON-LD processor encounters *null*, the associated entry or value is always removed when expanding the JSON-LD document.

However, in the context of a partial update or merge operation (see clauses 5.5.8 and 5.5.12), an NGSI-LD Null shall be used to indicate the removal of a target member. Thus, for representing deleted elements in notifications and in the temporal evolution, the URI *"urn:ngsi-ld:null"* is used as a Property *value* or Relationship *object* and the JSON object *{"@none": "urn:ngsi-ld:null"}* for the *languageMap* of a Language Property, respectively. In all other cases, implementations shall raise an error of type *BadRequestData* if an NGSI-LD Null value is encountered.

As *null* cannot be used as a value in JSON-LD, there is still the possibility of using a JSON null literal *{"@type": "@json", "@value": null}* instead. JSON literals are not to be expanded in JSON-LD and thus the respective element is not removed during JSON-LD expansion.

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

The use of URI in the context of the present document also includes the use of International Resource Identifiers (IRIs) as defined in IETF RFC 3987 [23], which extends the use of characters to Unicode characters [22] beyond the ASCII character set, enabling the support of languages other than English.

5.2.2 Common members

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

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

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

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

Name	Data Type	Restriction	Cardinality	Description
createdAt	string	*DateTime* (clause 4.6.3)	0..1	Entity creation timestamp. See clause 4.8
modifiedAt	string	*DateTime* (clause 4.6.3)	0..1	Entity last modification timestamp. See clause 4.8
deletedAt	string	*DateTime* (clause 4.6.3)	0..1	Entity deletion timestamp. See clause 4.8 It is only used in notifications reporting deletions and in the Temporal Representation of Entities (clause 4.5.6), Properties (clause 4.5.7), Relationships (clause 4.5.8) and LanguageProperties (clause 5.2.32) and VocabularyProperties (clause 5.2.35)

5.2.3 @context

When encoding NGSI-LD Entities, Context Source Registrations, Subscriptions and Notifications, as pure JSON-LD (MIME type "application/ld+json"), a user @context (as described in clause 4.4) shall be included as a special member of the corresponding JSON-LD Object. Table 5.2.3-1 gives a precise definition of this special member.

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

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

5.2.4 Entity

This type represents the data needed to define an NGSI-LD entity as mandated by clause 4.5.1.

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	String	Valid URI	1	Entity id
type	String or String[]		1	Entity Type(s). Both short hand string(s) (type name) or URI(s) are allowed
scope	String or String[]	See clause 4.18	0..1	Scope
location	GeoProperty	See datatype definition in clause 5.2.7	0..1	Default geospatial Property of an entity. See clause 4.7
observationSpace	GeoProperty	See datatype definition in clause 5.2.7	0..1	See clause 4.7
operationSpace	GeoProperty	See datatype definition in clause 5.2.7	0..1	See clause 4.7
<Property Name>	Property or Property[]	See datatype definitions in clauses 5.2.5, 5.2.7 and 5.2.32	0..N	Property as mandated by clause 4.5.1. For each Property identified by the same Property Name, there can be one or more instances
<Relationship Name>	Relationship or Relationship[]	See datatype definition in clause 5.2.6	0..N	Relationship as mandated by clause 4.5.2. For each Relationship identified by the same Relationship Name, there can be one or more instances

5.2.5 Property

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

The supported JSON members shall follow the requirements provided in table 5.2.5-1 below. The datatype definition defines all the required attributes for the normalized representation. In the concise representation, the Attribute "type" member can be omitted as type="Property" can be inferred from the presence of the "value" member. Furthermore, in the concise representation of a Property, the "value" member cannot be a GeoJSON Object (as defined in clause 4.7) as it would be interpreted as a GeoProperty (see clause 5.2.7).

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

Name	Data Type	Restriction	Cardinality	Description
type	String	It shall be equal to "Property"	1	Node type
value	Any JSON value as defined by IETF RFC 8259 [6]	See NGSI-LD Value definition in clause 3.1	1	Property Value
previousValue	Any JSON value as defined by IETF RFC 8259 [6]	Only used in Notifications, if the *showChanges* option is explicitly requested	0..1	Previous Property Value
observedAt	String	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
unitCode	String	As mandated by [15]	0..1	Property Value's unit code
datasetId	String	Valid URI	0..1	It allows identifying a set or group of property values
<Property Name>	Property or Property[]	See datatype definitions in clauses 5.2.5, 5.2.7 and 5.2.32	0..N	Properties of Property. For each Property identified by the same Property Name, there can be one or more instances
<Relationship Name>	Relationship or Relationship[]	See datatype definition in clause 5.2.6	0..N	Relationships of Property. For each Relationship identified by the same Relationship Name, there can be one or more instances

5.2.6 Relationship

This type represents the data needed to define a Relationship as mandated by clause 4.5.3.

The supported JSON members shall follow the requirements provided in table 5.2.6-1 below. The datatype definition defines all the required attributes for the normalized representation. In the concise representation, the Attribute "type" member can be omitted as type="Relationship" can be inferred from the presence of the "object" member.

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	String	Valid URI	1	Relationship's target object
previousObject	String	Valid URI. Only used in Notifications, if the *showChanges* option is explicitly requested	0..1	Previous Relationship's target object
observedAt	String	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
datasetId	String	Valid URI	0..1	It allows identifying a set or group of target relationship objects
<Property Name>	Property or Property[]	See datatype definitions in clauses 5.2.5, 5.2.7 and 5.2.32	0..N	Properties of the Relationship. For each Property identified by the same Property Name, there can be one or more instances
<Relationship Name>	Relationship or Relationship[]	See datatype definition in clause 5.2.6	0..N	Relationships of the Relationship. For each Relationship identified by the same Relationship Name, there can be one or more instances

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 below. The datatype definition defines all the required attributes for the normalized representation. In the concise representation, the Attribute "type" member can be omitted as "GeoProperty" can be inferred from the presence of the "value" member holding a GeoJSON Geometry as mandated by clause 4.7.

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]
previousValue	Any JSON value as defined by IETF RFC 8259 [6]	Only used in Notifications, if the *showChanges* option is explicitly requested	0..1	Previous GeoProperty Value
observedAt	String	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
datasetId	String	Valid URI	0..1	It allows identifying a set or group of property values
<Property Name>	Property or Property[]	See datatype definitions in clauses 5.2.5, 5.2.7 and 5.2.32	0..N	Properties of Property For each Property identified by the same Property Name, there can be one or more instances
<Relationship Name>	Relationship or Relationship[]	See datatype definition in clause 5.2.6	0..N	Relationships of Property. For each Relationship identified by the same Relationship Name, there can be one or more instances

5.2.8 EntityInfo

This type represents what Entities, Entity Types or group of Entity ids (as a regular expression pattern mandated by IEEE 1003.2™ [11]) can be provided (by Context Sources).

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

Notice that Cardinality of "type" being 1 implies that it is not possible to register what Entities can be provided by a Context Source just by their id or idPattern (i.e. without specifying their type).

Table 5.2.8-1: EntityInfo data type definition

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

5.2.9 CSourceRegistration

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

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

Table 5.2.9-1: CSourceRegistration data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI. Unique registration identifier. (JSON-LD @id).	0..1	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
type	String	It shall be equal to "ContextSourceRegistration"	1	JSON-LD @type Use reserved type for identifying Context Source Registration
registrationName	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. Empty array (0 length) is not allowed	1	Describes the Entities, Properties and Relationships for which the Context Source may be able to provide information
tenant	String		0..1	Identifies the tenant that has to be specified in all requests to the Context Source that are related to the information registered in this Context Source Registration. If not present, the default tenant is assumed. Should only be present in systems supporting multi-tenancy
observationInterval	TimeInterval	See data type definition in clause 5.2.11	0..1	If present, the Context Source can be queried for Temporal Entity Representations. (If latest Entity information is also provided, a separate Context Registration is needed for this purpose). The *observationInterval* specifies the time interval for which the Context Source can provide Entity information as specified by the *observedAt* Temporal Property. A temporal query based on the *observedAt* Temporal Property, which is the default, is matched against the *observationInterval* for overlap

Name	Data Type	Restriction	Cardinality	Description
managementInterval	TimeInterval	See data type definition in clause 5.2.11	0..1	If present, the Context Source can be queried for Temporal Entity Representations. (If latest Entity information is also provided, a separate Context Registration is needed for this purpose). The *managementInterval* specifies the time interval for which the Context Source can provide Entity information as specified by the *createdAt*, *modifiedAt* and *deletedAt* Temporal Properties. A temporal query based on the *createdAt*, *modifiedAt* or *deletedAt* Temporal Property is matched against the *managementInterval* for overlap
location	GeoJSON Geometry as mandated by clause 4.7		0..1	Location for which the Context Source may be able to provide information
observationSpace	GeoJSON Geometry as mandated by clause 4.7		0..1	Geographic location that includes the observation spaces of all entities as specified by their respective *observationSpace GeoProperty* for which the Context Source may be able to provide information
operationSpace	GeoJSON Geometry as mandated by clause 4.7		0..1	Geographic location that includes the operation spaces of all entities as specified by their respective *operationSpace GeoProperty* for which the Context Source may be able to provide information
expiresAt	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	String	It shall be a dereferenceable URI	1	Endpoint expressed as dereferenceable URI through which the Context Source exposes its NGSI-LD interface
contextSourceInfo	KeyValuePair[]		0..1	Generic {key, value} array to convey optional information to provide when contacting the registered Context Source
scope	String or String[]	Scope(s)	0..1	Scopes (see clause 4.18) for which the Context Source has Entities
mode	String	It shall be one of: "inclusive", "exclusive", "redirect" or "auxiliary" The mode is assumed to be "inclusive" if not explicitly defined	0..1	The definition of the mode of distributed operation (see clause 4.3.6) supported by the registered Context Source

Name	Data Type	Restriction	Cardinality	Description
operations	String[]	Entries are limited to the named API operations and named operation groups (see clause 4.20)	0..1	The definition limited subset of API operations supported by the registered Context Source If undefined, the default set of operations is "federationOps" (see clause 4.20)
refreshRate	String	String representing a duration in ISO 8601 format [17]	0..1	An indication of the likely period of time to elapse between updates at this registered endpoint. Brokers may optionally use this information to help implement caching.
management	Registration Management Info	See data type definition in clause 5.2.34	0..1	Holds additional optional registration management information that can be used to limit unnecessary distributed operation requests.
<CSource Property Name>	Any JSON value as defined by IETF RFC 8259 [6]		0..N	Each Context Source Property pertains to a characteristic of the Context Source the Context Source Registration describes

The members (defined by table 5.2.9-2) of the *CSourceRegistration* data structure are also defined. They are read-only and shall be automatically generated by NGSI-LD implementations. In the event that they are provided (in update or create operations) NGSI-LD implementations shall ignore them.

Table 5.2.9-2: Additional members of the CSourceRegistration data type

Name	Data Type	Restrictions	Cardinality	Description
status	string	Allowed values: "ok" "failed"	0..1	Read-only., Status of the Registration. It shall be "ok" if the last attempt to perform a distributed operation succeeded. It shall be "failed" if the last attempt to perform a distributed operation failed.
timesSent	Number	0 or greater value	0..1	Number of times that the registration triggered a distributed operation, including failed attempts.
timesFailed	Number	0 or greater value	0..1	Number of times that the registration triggered a distributed operation request that failed.
lastSuccess	string	DateTime (clause 4.6.3)	0..1	Timestamp corresponding to the instant when the last successfully distributed operation was sent. Created on first successful operation.
lastFailure	string	DateTime (clause 4.6.3)	0..1	Timestamp corresponding to the instant when the last distributed operation resulting in a failure (for instance, in the HTTP binding, an HTTP response code other than 2xx) was returned.

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 in clause 5.2.8. Empty array (0 length) is not allowed. Restrictions in clause 4.3.6 apply as well	0..1	Describes the entities for which the CSource may be able to provide information
propertyNames	string []	Property Names as short-hand strings. Empty array is not allowed. Restrictions in clause 4.3.6 apply as well	0..1	Describes the Properties that the CSource may be able to provide
relationshipNames	string []	Relationship Names as short-hand strings. Empty array is not allowed. Restrictions in clause 4.3.6 apply as well	0..1	Describes the Relationships that the CSource may be able to provide

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

5.2.11 TimeInterval

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

Table 5.2.11-1: TimeInterval data type definition

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

5.2.12 Subscription

This datatype represents a Context Subscription.

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

Table 5.2.12-1: Subscription data type definition

Name	Data Type	Restrictions	Cardinality	Description
id	String	Valid URI	0..1	Subscription identifier (JSON-LD @id). 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
type	String	It shall be equal to "Subscription"	1	JSON-LD @type
subscriptionName	String		0..1	A (short) name given to this Subscription
description	String		0..1	Subscription description
entities	EntitySelector[]	See data type definition in clause 5.2.33. Empty array (0 length) is not allowed	0..1	Entities subscribed

Name	Data Type	Restrictions	Cardinality	Description
watchedAttributes	String[]	Attribute Name as short-hand string. if *timeInterval* is present it shall not appear (0 cardinality). Empty array (0 length) is not allowed	0..1	Watched Attributes (Properties or Relationships). If not defined it means any Attribute
notificationTrigger	String[]	Valid notification triggers are *entityCreated*, *entityUpdated*, *entityDeleted*, *attributeCreated*, *attributeUpdated*, *attributeDeleted*	0..1	The notification triggers listed indicate what kind of changes shall trigger a notification. If not present, the default is the combination *attributeCreated* and *attributeUpdated*. *entityUpdated* is equivalent to the combination *attributeCreated*, *attributeUpdated* and *attributeDeleted*
timeInterval	Number	Greater than 0 if *watchedAttributes* is present it shall not appear (0 cardinality)	0..1	Indicates that a notification shall be delivered periodically regardless of attribute changes. Actually, when the time interval (in seconds) specified in this value field is reached
q	String	A valid query string as per clause 4.9	0..1	Query that shall be met by subscribed entities in order to trigger the notification
geoQ	GeoQuery	See data type definition in clause 5.2.13	0..1	Geoquery that shall be met by subscribed entities in order to trigger the notification
csf	String	A valid query string as per clause 4.9	0..1	Context source filter that shall be met by Context Source Registrations describing Context Sources to be used for Entity Subscriptions
isActive	Boolean	*true* by default	0..1	Allows clients to temporarily pause the subscription by making it inactive. *true* indicates that the Subscription is under operation. *false* indicates that the subscription is paused and notifications shall not be delivered
notification	NotificationParams	See data type definition in clause 5.2.14	1	Notification details
expiresAt	String	*DateTime* (see clause 4.6.3)	0..1	Expiration date for the subscription
throttling	Number	Greater than 0. If *timeInterval* is present it shall not appear (0 cardinality)	0..1	Minimal period of time in seconds which shall elapse between two consecutive notifications
temporalQ	TemporalQuery	See data type definition in clause 5.2.21	0..1	Temporal Query to be used *only in Context Registration Subscriptions* for matching Context Source Registrations of Context Sources providing temporal information
scopeQ	String	See clause 4.19	0..1	Scope query
lang	String	A natural language filter in the form of a IETF RFC 5646 [28] language code	0..1	Language filter to be applied to the query (clause 4.15)
jsonldContext	String	Dereferenceable URI		The dereferenceable URI of the JSON-LD @context to be used when sending a notification resulting from the subscription. If not provided, the @context used for the subscription shall be used as a default

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

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

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

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

5.2.13 GeoQuery

This datatype represents a geoquery used for Subscriptions.

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

Table 5.2.13-1: GeoQuery data type definition

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

5.2.14 NotificationParams

5.2.14.1 NotificationParams data type definition

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

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

Table 5.2.14.1-1: NotificationParams data type definition

Name	Data Type	Restrictions	Cardinality	Description
attributes	string[]	Attribute name as short-hand string. Empty array (0 length) is not allowed	0..1	Attribute Names (Properties or Relationships) to be included in the notification payload body. If undefined it will mean all Attributes
sysAttrs	boolean	*false* by default	0..1	If *true*, the system generated attributes *createdAt* and *modifiedAt* are included in the response payload body, in the case of a deletion also *deletedAt*
format	string	It shall be one of: "normalized", "concise", "keyValues" (or its synonym "simplified")	0..1	Conveys the representation format of the entities delivered at notification time. By default, it will be in the normalized format

Name	Data Type	Restrictions	Cardinality	Description
showChanges	boolean	*false* by default	0..1	If *true* the previous *value* (*previousValue)* of Properties or *languageMap* (*previousLanguageMap)* of Language Properties or *object* (*previousObject*) of Relationships is provided in addition to the current one. This requires that it exists, i.e. in case of modifications and deletions, but not in the case of creations. showChanges cannot be *true* in case format is "keyValues"
endpoint	Endpoint	See data type definition in clause 5.2.15	1	Notification endpoint details
status	string	Allowed values: "ok", "failed"	0..1	Status of the Notification. It shall be "ok" if the last attempt to notify the subscriber succeeded. It shall be "failed" if the last attempt to notify the subscriber failed

5.2.14.2 Additional members

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

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

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

Name	Data Type	Restrictions	Cardinality	Description
timesSent	Number	Greater than 0	0..1	Number of times that the notification has been sent. Provided by the system when querying the details of a subscription
timesFailed	Number	Greater than 0	0..1	Number of times an unsuccessful response (or timeout) has been received when notified the notification. 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 has been 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) has been 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 has been 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. This can include, in addition the endpoint's URI, a generic{key, value} array, named *receiverInfo*, which contains, in a generalized form, whatever extra information the broker shall convey to the receiver in order for the broker to successfully communicate with receiver (e.g. Authorization material), or for the receiver to correctly interpret the received content (e.g. the Link URL to fetch an @context). Additionally, it can include another generic{key, value} array, named *notifierInfo*, which contains the configuration that the broker needs to know in order to correctly set up the communication channel towards the receiver (e.g. MQTT-Version, MQTT-QoS, in case of MQTT binding, as defined in clause 7.2).

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	String	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" "application/geo+json"	0..1	Intended to convey the MIME type of the notification payload body (JSON, or JSON-LD, or GeoJSON). If not present, default is "application/json".
timeout	Number	Greater than 0	0..1	Maximum period of time in milliseconds which may elapse before a notification is assumed to have failed. The NGSI-LD system can override this value. This only applies if the binding protocol always returns a response.
cooldown	Number	Greater than 0	0..1	Once a failure has occurred, minimum period of time in milliseconds which shall elapse before attempting to make a subsequent notification to the same endpoint after failure. If requests are received before the cooldown period has expired, no notification is sent.
receiverInfo	KeyValuePair[]		0..1	Generic {key, value} array to convey optional information to the receiver.
notifierInfo	KeyValuePair[]		0..1	Generic {key, value} array to set up the communication channel.

5.2.16 BatchOperationResult

This datatype represents the result of a batch operation.

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

Table 5.2.16-1: BatchOperationResult data type definition

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

5.2.17 BatchEntityError

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

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

Table 5.2.17-1: BatchEntityError data type definition

Name	Data Type	Restrictions	Cardinality	Description
entityId	String	Valid URI	1	Entity Id corresponding to the Entity in error
registrationId	String	Valid URI	0..1	Registration Id corresponding to a failed distributed operation (optional)
error	ProblemDetails (see IETF RFC 7807 [10])		1	One instance per Entity in error

5.2.18 UpdateResult

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

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

Table 5.2.18-1: UpdateResult data type definition

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

5.2.19 NotUpdatedDetails

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

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

Table 5.2.19-1: NotUpdatedDetails data type definition

Name	Data Type	Restrictions	Cardinality	Description
attributeName	String		1	Attribute name
reason	String		1	Reason for not having changed such Attribute
registrationId	String	Valid URI	0..1	Registration Id corresponding to a failed distributed operation (optional)

5.2.20 EntityTemporal

This is the same data type as mandated by clause 5.2.4 with the only deviation that the representation of Properties and Relationships shall be the temporal one (arrays of (Property or Relationship) instances represented by JSON-LD objects) as defined in clauses 4.5.7 and 4.5.8. Alternatively it is possible to specify the EntityTemporal by using the "Simplified Temporal Representation of an Entity", as defined in clause 4.5.9.

5.2.21 TemporalQuery

This datatype represents a temporal query.

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

Table 5.2.21-1: TemporalQuery data type definition

Name	Data Type	Restrictions	Cardinality	Description
timerel	String	Allowed values: "before", "after" and "between"	1	Represents the temporal relationship as defined by clause 4.11
timeAt	String representing the *timeAt* parameter as defined by clause 4.11	It shall be a *DateTime*	1	
endTimeAt	String representing the *endTimeAt* parameter as defined by clause 4.11	It shall be a *DateTime.* Cardinality shall be 1 if *timerel* is equal to "between"	0..1	
timeproperty	String representing a Temporal Property name	Allowed values: "observedAt", "createdAt", "modifiedAt" and "deletedAt". If not specified, the default is "observedAt". (See clause 4.8)	0..1	

5.2.22 KeyValuePair

This datatype represents the optional information that is required when contacting an endpoint for notifications.

The supported members shall follow the indications provided in table 5.2.22-1. They are intended to represent a key/value pair.

Example optional information includes additional HTTP Headers such as:

- The HTTP Authentication Header.
- The HTTP Prefer Header (IETF RFC 7240 [26]) used when notifying the GeoJSON Endpoint.

Table 5.2.22-1: KeyValuePair data type definition

Name	Data Type	Restrictions	Cardinality	Description
key	String	Binding-dependent	1	The key of the key/value pair
value	String	Binding-dependent	1	The value of the key/value pair

5.2.23 Query

This datatype represents the information that is required in order to convey a query when a "Query Entities" operation or a "Query Temporal Evolution of Entities" operation is to be performed (as per clauses 5.7.2 and 5.7.4, respectively).

The supported JSON members shall follow the requirements provided in table 5.2.23-1.

Table 5.2.23-1: Query data type definition

Name	Data Type	Restrictions	Cardinality	Description
type	string	It shall be equal to "Query"	1	JSON-LD @type
entities	EntitySelector[]	See data type definition in clause 5.2.33. Empty array (0 length) is not allowed	0..1	Entity ids, id pattern and Entity types that shall be matched by Entities in order to be retrieved
attrs	string[]	Attribute Name as short-hand string. Empty array (0 length) is not allowed	0..1	List of Attributes that shall be matched by Entities in order to be retrieved. If not present all Attributes will be retrieved
q	string	A valid query string as per clause 4.9	0..1	Query that shall be matched by Entities in order to be retrieved
geoQ	GeoQuery	See data type definition in clause 5.2.13	0..1	Geoquery that shall be matched by Entities in order be retrieved
csf	string	A valid query string as per clause 4.9	0..1	Context source filter that shall be matched by Context Source Registrations describing Context Sources to be used for retrieving Entities
temporalQ	TemporalQuery	See data type definition in clause 5.2.21	0..1	Temporal Query to be present only for "Query Temporal Evolution of Entities" operation (clause 5.7.4)
scopeQ	String	See clause 4.19	0..1	Scope query
lang	string	A natural language filter in the form of a IETF RFC 5646 [28] language code	0..1	Language filter to be applied to the query (clause 4.15)

5.2.24 EntityTypeList

This type represents the data needed to define the entity type list representation as mandated by clause 4.5.10.

The supported JSON members shall follow the requirements provided in table 5.2.24-1.

Table 5.2.24-1: NGSI-LD EntityTypeList data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI	1	URI that is unique within the system scope. Identifier for the entity type list
type	String	It shall be equal to "EntityTypeList"	1	JSON-LD @type
typeList	String[]		1	List containing the entity type names

5.2.25 EntityType

This type represents the data needed to define the elements of the detailed entity type list representation as mandated by clause 4.5.11.

The supported JSON members shall follow the requirements provided in table 5.2.25-1.

Table 5.2.25-1: NGSI-LD EntityType data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI	1	Fully Qualified Name (FQN) of the entity type being described
type	String	It shall be equal to "EntityType"	1	JSON-LD @type
typeName	String		1	Name of the entity type, short name if contained in @context
attributeNames	String[]		1	List containing the names of attributes that instances of the entity type can have

5.2.26 EntityTypeInfo

This type represents the data needed to define the detailed entity type information representation as mandated by clause 4.5.12.

The supported JSON members shall follow the requirements provided in table 5.2.26-1.

Table 5.2.26-1: NGSI-LD EntityTypeInfo data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI	1	Fully Qualified Name (FQN) of the entity type being described
type	String	It shall be equal to "EntityTypeInfo"	1	JSON-LD @type
typeName	String		1	Name of the entity type, short name if contained in @context
entityCount	Number	Unsigned integer	1	Number of entity instances of this entity type
attributeDetails	Attribute[]	See data type definition in clause 5.2.28. Attribute with only the elements "id", "type", "attributeName" and "attributeTypes"	1	List of attributes that entity instances with the specified entity type can have

5.2.27 AttributeList

This type represents the data needed to define the attribute list representation as mandated by clause 4.5.13.

The supported JSON members shall follow the requirements provided in table 5.2.27-1.

Table 5.2.27-1: NGSI-LD AttributeList data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI	1	URI that is unique within the system scope. Identifier for the attribute list
type	String	It shall be equal to "AttributeList"	1	JSON-LD @type
attributeList	String[]		1	List containing the attribute names

5.2.28 Attribute

This type represents the data needed to define the attribute information needed as:

- part of the entity type information representation as mandated by clause 4.5.12;
- the detailed attribute list representation as mandated by clause 4.5.14;
- the attribute information representation as mandated by clause 4.5.15.

The supported JSON members shall follow the requirements provided in table 5.2.28-1.

Table 5.2.28-1: NGSI-LD Attribute data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI	1	Full URI of attribute name
type	String	It shall be equal to "Attribute"	1	JSON-LD @type
attributeName	String		1	Name of the attribute, short name if contained in @context
attributeCount	Number	Unsigned integer	0..1	Number of attribute instances with this attribute name
attributeTypes	String[]		0..1	List of attribute types (e.g. Property, Relationship, GeoProperty) for which entity instances exist, which contain an attribute with this name
typeNames	String[]		0..1	List of entity type names for which entity instances exist containing attributes that have the respective name

5.2.29 Feature

This data type represents a spatially bounded Entity in GeoJSON format, as mandated by IETF RFC 7946 [8]. The supported JSON members shall follow the requirements provided in table 5.2.29-1.

Table 5.2.29-1: Feature data type definition

Name	Data Type	Restriction	Cardinality	Description
id	String	Valid URI	1	Entity id
type	String	It shall be equal to "Feature"	1	GeoJSON Type
geometry	GeoJSON Object	The value field from the matching GeoProperty (as specified in clause 4.5.16) or null	1	Null if no matching GeoProperty
properties	FeatureProperties	See data type definition	1	List of attributes as mandated by clause 5.2.31
@context	URI, JSON Object, or JSON Array	See [2], section 5.1	0..1	JSON-LD @context. This field is only present if requested in the payload by the HTTP Prefer Header (IETF RFC 7240 [26])

5.2.30 FeatureCollection

This data type represents a list of spatially bounded Entities in GeoJSON format, as mandated by IETF RFC 7946 [8]. The supported JSON members shall follow the requirements provided in table 5.2.30-1.

Table 5.2.30-1: FeatureCollection data type definition

Name	Data Type	Restriction	Cardinality	Description
type	String	It shall be equal to "FeatureCollection"	1	GeoJSON Type
features	Feature[]	See data type definition	1..N	In the case that no matches are found, "features" will be an empty array
@context	URI, JSON Object, or JSON Array	See [2], section 5.1	0..1	JSON-LD @context. This field is only present if requested in the payload by the HTTP Prefer Header (IETF RFC 7240 [26])

5.2.31 FeatureProperties

This data type represents the type and the associated attributes (Properties and Relationships) of an Entity in GeoJSON format.

Table 5.2.31-1: NGSI-LD Entity data type definition

Name	Data Type	Restriction	Cardinality	Description
type	String or String[]	Entity Type	1	Entity Type (or JSON array, in case of Entities with multiple Entity Types). Both short hand string (type name) or URI are allowed.
<Property Name>	Property or Property[]	See data type definition	0..N	Property as mandated by clause 4.5.1. For each Property identified by the same Property Name, there can be one or more instances.
<Relationship Name>	Relationship or Relationship []	See data type definition	0..N	Relationship as mandated by clause 4.5.2. For each Relationship identified by the same Relationship Name, there can be one or more instances.

5.2.32 LanguageProperty

This type represents the data needed to define a LanguageProperty as mandated by clause 4.5.18. Note that in case of concise representation, the type can be omitted (see clause 4.5.18.3).

The supported JSON members shall follow the requirements provided in table 5.2.32-1.

Table 5.2.32-1: NGSI-LD LanguageProperty data type definition

Name	Data Type	Restriction	Cardinality	Description
type	string	It shall be equal to "LanguageProperty"	1	Node type
languageMap	JSON object	A set of key-value pairs whose keys shall be strings representing IETF RFC 5646 [28] language codes and whose values shall be JSON strings or arrays of JSON strings	1	String Property Values defined in multiple natural languages
previousLanguageMap	JSON object	A set of key-value pairs whose keys shall be strings representing IETF RFC 5646 [28] language codes and whose values shall be JSON strings. Only used in Notifications, if the *showChanges* option is explicitly requested	0..1	Previous Language Property's languageMap
observedAt	String	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
datasetId	String	Valid URI	0..1	It allows identifying a set or group of property values
<Property Name>	Property or Property[]	See datatype definitions in clauses 5.2.5, 5.2.7 and 5.2.32	0..N	Properties of Property. For each Property identified by the same Property Name, there can be one or more instances

Name	Data Type	Restriction	Cardinality	Description
<Relationship Name>	Relationship or Relationship[]	See datatype definition in clause 5.2.6	0..N	Relationships of Property. For each Relationship identified by the same Relationship Name, there can be one or more instances

5.2.33 EntitySelector

This type selects which entity or group of entities are queried or subscribed to by Context Consumers. Entities can be specified by their id, Entity Types or group of Entity ids (as a regular expression pattern mandated by IEEE POSIX 1003.2™ [11]).

The JSON members shall follow the indications provided in table 5.2.33-1. *id* takes precedence over *idPattern*.

Table 5.2.33-1: EntitySelector data type definition

Name	Data Type	Restrictions	Cardinality	Description
id	String	Valid URI	0..1	Entity identifier
idPattern	String	Regular expression as per IEEE POSIX 1003.2™ [11]	0..1	A regular expression which denotes a pattern that shall be matched by the provided or subscribed Entities
type	String	A valid type selection string as per clause 4.17	1	Selector of Entity Type(s)

5.2.34 RegistrationManagementInfo

This type represents the data to alter the default behaviour of a Context Broker when making a distributed operation request to a registered Context Source. The supported JSON members shall follow the indications provided in table 5.2.34-1. Brokers may override these recommendations.

Table 5.2.34-1: RegistrationManagementInfo data type definition

Name	Data Type	Restrictions	Cardinality	Description
localOnly	Boolean		0..1	If **localOnly=true** then distributed operations associated to this Context Source Registration will act only on data held directly by the registered Context Source itself (see clause 4.3.6.4).
cacheDuration	String	String representing a duration in ISO 8601 format [17]	0..1	Minimal period of time which shall elapse between two consecutive context information consumption operations (as defined in clause 5.7) related to the same context data will occur. If the cacheDuration latency period has not been reached, a cached value for the entity or its attributes shall be returned where available.
timeout	Number	Greater than 0	0..1	Maximum period of time in milliseconds which may elapse before a forwarded request is assumed to have failed.
cooldown	Number	Greater than 0	0..1	Minimum period of time in milliseconds which shall elapse before attempting to make a subsequent forwarded request to the same endpoint after failure. If requests are received before the cooldown period has expired, a timeout error response for the registration is automatically returned.

5.2.35 VocabularyProperty

This type represents the data needed to define a VocabularyProperty as mandated by clause 4.5.20.In case of concise representation, the type can be omitted (see clause 4.5.20.3).

The supported JSON members shall follow the requirements provided in table 5.2.35-1.

Table 5.2.35-1: NGSI-LD VocabularyProperty data type definition

Name	Data Type	Restriction	Cardinality	Description
type	String	It shall be equal to "VocabularyProperty"	1	Node type
vocab	String or string[]		1	String Values which shall be type coerced to URIs based on the supplied @context
previousVocab	String or String[]	Only used in Notifications, if the *showChanges* option is explicitly requested	0..1	Previous Vocabulary Property's vocab
observedAt	String	*DateTime* (clause 4.6.3)	0..1	Timestamp. See clause 4.8
datasetId	String	Valid URI	0..1	It allows identifying a set or group of property values
<Property Name>	Property or Property[]	See datatype definitions in clauses 5.2.5, 5.2.7 and 5.2.32	0..N	Properties of Property. For each Property identified by the same Property Name, there can be one or more instances
<Relationship Name>	Relationship or Relationship[]	See datatype definition in clause 5.2.6	0..N	Relationships of Property. For each Relationship identified by the same Relationship Name, there can be one or more instances

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	String	Valid 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	String	Valid 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[] or FeatureCollection		1	The content of the notification as NGSI-LD Entities. See clause 5.2.4. If the notification has been triggered from a Subscription that has the `notification.endpoint.accept` field set to `application/geo+json` then `data` is returned as a FeatureCollection. In this case, if the `notification.endpoint.receiverInfo` contains the key "Prefer" and it is set to the value "body=json", then the FeatureCollection will not contain an @context field. If `endpoint.accept` is not set or holds another value then Entity[] is returned.

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 table 5.3.2-1.

Table 5.3.2-1: CSourceNotification data type definition

Name	Data Type	Restrictions	Cardinality	Description
id	String	Valid URI	1	CSource notification identifier (JSON-LD @id)
type	String	It shall be equal to "ContextSourceNotification"	1	JSON-LD @type
subscriptionId	String	Valid 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 CSourceRegistrations. See clause 5.2.9
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, Attributes, Context Source Registrations or Subscriptions) it is necessary to have a means of describing a set of modifications to their content.

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

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

- *id* (optional for certain bindings where it can be determined from the operation signature). It shall be equal to the id of the target (mutated) NGSI-LD element. Attribute Fragments do not contain explicit ids.
- *type* (optional for certain bindings where it can be determined from the operation signature). It shall contain the Type Name(s) 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.

EXAMPLE 1: The following Subscription Fragment allows the modification of a Subscription by changing its endpoint's URI:

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

- A member (following the same data representation and nesting structure) with value equal to an NGSI-LD Null shall cause for the member to be removed from the target NGSI-LD element.

EXAMPLE 2: The following NGSI-LD Fragment allows the modification of an Entity by changing its "batteryLevel" Attribute, updating the "observedAt" sub-Attribute, removing the "providedBy" sub-Attribute and removing the "uncharged" Attribute from the Entity:

```
{
    "id": "urn:ngsi-ld:TemperatureSensor:001",
    "type": "TemperatureSensor",
    "batteryLevel": {
        "type": "Property",
        "value": 7,
        "observedAt": "2022-03-14T12:51:02.000Z",
        "providedBy": "urn:ngsi-ld:null"
    },
    "uncharged" : {
        "type": "Property",
        "value": "urn:ngsi-ld:null"
```

```
            }
        }
```

5.5 Common Behaviours

5.5.1 Introduction

This clause defines common behaviours for the API operations.

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

5.5.2 Error types

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

Table 5.5.2-1: Error types in NGSI-LD

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

5.5.3 Error response payload body

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

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

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

5.5.4 General NGSI-LD validation

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

- If the request payload body is not a valid JSON document then an error of type *InvalidRequest* shall be raised.

- If the data included by the JSON-LD document is not syntactically correct, according to the @context or the API data type definitions, then an error of type *BadRequestData* shall be raised.

- Any attempt to use *"urn:ngsi-ld:null"* as a first level member value (*"<key>":"urn:ngsi-ld:null")*, with the exception of NGSI-LD Fragments (see clause 5.4) used in partial update and merge operations (as mandated by clause 5.5.8 and clause 5.5.12) or to represent deleted Properties in concise representation as part of notifications, shall result in an error of type *BadRequestData*.

- Any attempt to use *"urn:ngsi-ld:null"* as the right-hand side of *value* in a Property, as the right-hand side of *object* in a Relationship or to use *{"@none": "urn:ngsi-ld:null"}* as the right-hand side of *languageMap*, with the exception of NGSI-LD Fragments (see clause 5.4) used in update and merge operations (as mandated by clause 5.5.8 and clause 5.5.12) and the representation of deleted Properties, Relationships or Language Properties in notifications and the temporal evolution, shall result in an error of type *BadRequestData*.

- Any attempt to use *"urn:ngsi-ld:null"* as the value of a key value pair within a JSON object, which is the right-hand side of the *value* of a Property, with the exception of NGSI-LD Fragments used in merge operations (see clause 5.5.12), shall result in an error of type *BadRequestData*.

5.5.5 Default @context assignment

If the input provided by an API client does not include any @context, then the implementation shall at minimum assign the Core @context to such an input. In addition, the Context Broker implementation may allow configuring a default user @context (with default terms), to be used when no user @context is provided. The Core @context shall always take precedence.

5.5.6 Operation execution

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

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

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

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

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

5.5.7 Term to URI expansion or compaction

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

The term to URI expansion or compaction shall be performed using a @context as described by the JSON-LD specification [2] (section 5.1), and in clause 4.4. **In the absence of a user @context, the term expansion or compaction shall be performed according to clause 5.5.5**.

For the avoidance of doubt, the @context used to perform compaction or expansion of terms **shall be the one provided by each API call** (or the default @context in its absence), and not any other @context which might have been supplied previously. For instance, when performing "Query Entity" operations (clause 5.7.2), the @context used to perform URI expansion and compaction shall be the one provided by the request.

In case of HTTP binding via GET (clause 6.4.3.2) of the "Query Entity" operation, this means using the JSON-LD Link Header as described by the JSON-LD specification [2], section 6.2. In case of HTTP binding via POST (clause 6.23.3.1), of the "Query Entity" operation, this means giving the @context either via Link Header or within the payload body, depending on the Content-Type Header being application/json or application/ld+json, respectively.

It is important to warn users that updating a @context could lead to behaviour that might be perceived as inconsistent. If, for instance, a fully qualified name that qualified a given short-hand name is changed, from that moment onwards, the short-hand name is referencing a different Attribute. This will effectively change the results of queries that use the given short name, possibly not giving back anymore the expected set of results.

Moreover, this user @context shall not:

- Contain JSON-LD Scoped Contexts (see [2], section 4.1.8).
- Be embedded into NGSI-LD Attributes, i.e. there cannot be parts of the user @context other than at the top level of the NGSI-LD document.

Parts of user @context that are not following the two points above should result in an error of type *BadRequestData*, because JSON-LD Scoped Contexts and nested embedded @context could be used to modify terms defined in the Core @context or to reshape NGSI-LD Elements during the expansion of terms.

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

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

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

5.5.8 Partial Update Patch Behaviour

The Partial Update Patch procedure modifies an existing NGSI-LD element by overwriting the data at the Attribute level, replacing it with the data provided in the NGSI-LD Fragment.

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.

With respect to update operations, implementations shall perform an algorithm equivalent to the one described below (adapted from IETF RFC 7396 [16]), in order to observe the name to URI expansion rules and the JSON-LD *null* processing):

- For each member of the Fragment perform the term to URI expansion.
- If the provided Fragment (a JSON Merge Patch document) 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 contained by the target, the target member value is replaced by the value given in the Fragment. In the case of a member representing a reified Property or Relationship including a *datasetId*, such member is only replaced if the *datasetId* is the same, otherwise the member of the Fragment is added as a new instance to the target. If no *datasetId* is present, the default Attribute instance is targeted and replaced if present and otherwise added. In case of a member *type* (of an entity) in Entity Fragments, all included Entity Types are added, if they are not already contained in the *type* member of the target.

- For each member of the Fragment, whose value is an NGSI-LD Null, contained by the target, the target member is deleted. In the case of deleting a specific Attribute instance with a *datasetId*, the handling shall be in accordance with the description found in clause 5.6.5. A *datasetId* cannot be deleted by setting it to the value *"urn:ngsi-ld:null"*.

EXAMPLE 1: Given an Entity containing the following Property:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 25,
    "unitCode": "CEL"
    "observedAt": "2022-03-14T01:59:26.535Z"
  }
}
```

Applying **partial attribute update** operation (as defined in clause 5.6.4) at the Attribute level onto the "temperature" Attribute, with the following Attribute Fragment payload:

```
{
  "type" : "Property",
  "value" : 100,
  "observedAt": "2022-03-14T13:00:00.000Z"
}
```

Results in an overwrite of the "value" and "observedAt" sub-Attributes, leaving the "unitCode" sub-Attribute untouched as shown:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 100,
    "unitCode": "CEL"
    "observedAt": "2022-03-14T13:00:00.000Z"
  }
}
```

EXAMPLE 2: Given an Entity containing the following Property:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 25,
    "unitCode": "CEL"
    "observedAt": "2022-03-14T01:59:26.535Z"
  }
}
```

Applying an **update attributes** operation (as defined in clause 5.6.2) onto the Entity as a whole with the following Entity Fragment payload:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 100,
    "observedAt": "2022-03-14T13:00:00.000Z"
  }
}
```

Results in an overwrite of the whole "temperature" Attribute – other Attributes would remain untouched. The result is that the "value" and "observedAt" sub-Attributes are updated and the "unitCode" sub-Attribute is removed as shown:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 100,
    "observedAt": "2022-03-14T13:00:00.000Z"
  }
}
```

EXAMPLE 3: Given an Entity containing the following Property:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 25,
    "unitCode": "CEL"
    "observedAt": "2022-03-14T01:59:26.535Z"
  }
}
```

Applying an **update attributes** operation (as defined in clause 5.6.2) onto the Entity as a whole, with the following Entity Fragment payload:

```
{
  "temperature": {
    "type" : "Property",
    "value" : "urn:ngsi-ld:null"
  }
}
```

Results in the deletion of the whole "temperature" Attribute – all other Attributes remain untouched.

5.5.9 Pagination Behaviour

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

- Let **Md** be equal to the default maximum number of NGSI-LD Elements to be retrieved by the API during each query pagination iteration, as defined by the NGSI-LD implementation.
- Let **Mc** be equal to the maximum number of NGSI-LD Elements to be retrieved as requested by the NGSI-LD Client. If **Mc** is undefined then it shall be equal to **Md**.
- Let **L** be the maximum number of NGSI-LD Elements to be retrieved by the API during each query pagination iteration. **L** shall be equal to **Mc**.
- During query execution and for each pagination iteration, the query resolution mechanisms of the NGSI-LD System shall ensure that only up to a maximum of **L** NGSI-LD Elements are retrieved and returned to the NGSI-LD client, i.e. the maximum page size per iteration shall not overpass **L**. Nonetheless, implementations shall take care of not overpassing a maximum size of response payload body, which, in practice, implies that, under certain circumstances, the number of Elements retrieved per page can be lower than **L**.
- After the retrieval of each page (containing at most **L NGSI-LD Elements**) implementations shall check whether there are pending NGSI-LD Elements to be retrieved in the context of the current query. If that is the case, implementations shall flag NGSI-LD Clients of the existence of such NGSI-LD Elements. Ultimately, the flagging mechanisms used shall depend on each API binding but shall be present as mandated by the present clause.
- When flagging the existence of additional NGSI-LD Elements (pages) pending to be retrieved, generally, implementations shall provide NGSI-LD Clients pointers to get access to both the following page of NGSI-LD Elements and the previous one, according to the current pagination iteration.
- The pointer to the previous page of NGSI-LD Elements shall be included for all pagination iterations excepting the first one, as, obviously, there will be no previous NGSI-LD Elements.
- When the last page of NGSI-LD Elements is reached, only the pointer to the previous page shall be provided to NGSI-LD Clients, so that they can detect that no more NGSI-LD Elements are available.
- The pointers to NGSI-LD Elements shall contain all the parameters needed to allow NGSI-LD Clients to retrieve the next and previous page, without further interactions with the API.

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

5.5.10 Multi-Tenant Behaviour

If a tenant is specified for an NGSI-LD operation, the operation shall only be applied to information related to the specified tenant. If no tenant is specified, the operation shall apply to the implicitly existing default tenant. If a tenant is explicitly specified, but the system implementing the NGSI-LD API does not support multi-tenancy, an error of type *NoMultiTenantSupport* should be raised.

In case an operation applies to a tenant, this information shall also be provided in the response to the operation. This also applies to notifications sent as a result of subscriptions (clauses 5.8 and 5.11).

A tenant is represented in form of a String. How the tenant is specified for an API operation is protocol binding specific. How tenants are created, is implementation-specific.

One implementation option is to support the implicit creation of tenants. This means that a tenant is implicitly created when an NGSI-LD operation for creating information targets a new tenant; this is the case for:

- Create Entity (clause 5.6.1).
- Batch Entity Creation (clause 5.6.7).
- Create or Update Temporal Representation of an Entity (clause 5.6.11).
- Create Subscription (clause 5.8.1).
- Register Context Source (clause 5.9.2).
- Create Context Source Registration (clause 5.11.2).

All other NGSI-LD operations, e.g. for retrieving, updating, appending or deleting information that target a non-existing tenant should raise an error of type *NonexistentTenant*.

If the system implementing the NGSI-LD API does not support multiple tenants, the attempt to register a Context Source with tenant information in the Context Source Registration should also result in an error of type *NoMultiTenantSupport*.

5.5.11 More than one instance of the same Entity in an Entity array

5.5.11.0 Foreword

The following operations operate on an array of entities (as input payload):

- Batch Entity Creation (clause 5.6.7)
- Batch Entity Creation or Update (Upsert) (clause 5.6.8)
- Batch Entity Update (clause 5.6.9)
- Batch Entity Delete (clause 5.6.10)
- Batch Entity Merge (clause 5.6.20)

It is allowed for such an input Entity array to contain more than one instance of the same entity (those instances have identical ids).

In order for such a request to be correctly handled, those instances that have the same id are processed by the Broker in the order they have in the array: the higher the index in the array, the later it will be processed. If the order is altered, the outcome may be altered.

All Entities and Attributes in the batch will get the same "modifiedAt" timestamp, so it makes sense to distinguish them via the "observedAt" temporal property.

Implementations shall treat the entity instances as if they had all arrived in separate requests.

The following clauses specify the behaviour in each case.

5.5.11.1 Batch Entity Creation case

The first occurrence of an entity in the input array (the oldest one) is used for the creation of the entity. Any subsequent instance of the same entity is reported as an error (entity already exists) in the response.

5.5.11.2 Batch Entity Creation or Update (Upsert) case

This operation has two modes of operation, with an optional flag to select between the two. The default behaviour is to replace any already existing entities, while the optional behaviour is to update already existing entities. Non existing entities are created in both modes.

If the entity does not yet exist, the first occurrence of an entity is used to create the entity, and subsequent instances of that same entity are used to either replace (default behaviour) or to update (optional behaviour) the entity. These replace or update operations shall be done **in chronological order**.

Only the entity resulting from merging all of the entity instances, in the correct order, is maintained in the current state (as defined in clause 4.3.1). For temporal evolution (as defined in clause 4.3.1) of Entities, all entity instances shall be taken into account, in the correct order.

5.5.11.3 Batch Entity Update case

This operation has two modes of operation, with an optional flag to select between the two. The default behaviour is to replace any already existing attributes of the entities, while the optional behaviour is to preserve already existing attributes of the entities.

Brokers shall send separate notifications for each individual update, taking throttling into account.

5.5.11.4 Batch Entity Delete case

The Batch Entity Delete operation has as input an array of Entity IDs, for the entities to be deleted. If an Entity ID is replicated in the array, the first occurrence will delete the entity, while subsequent occurrences of the same Entity ID will provoke an error in the response (entity does not exist).

5.5.11.5 Batch Entity Merge case

The Batch Entity Merge operation has as input an array of Entity IDs, for the entities to be merged. If an Entity ID is replicated in the array, these merge operations shall be done **in chronological order**. Only the entity resulting from merging all of the entity instances, in the correct order, is maintained in the current state (as defined in clause 4.3.1). For temporal evolution (as defined in clause 4.3.1) of Entities, all entity instances shall be taken into account, in the correct order.

5.5.12 Merge Patch Behaviour

The merge patch procedure modifies an existing NGSI-LD element by applying the set of changes found in an NGSI-LD Fragment data to the target resource. Unlike the partial update patch behaviour (described in clause 5.5.8), which replaces the complete element on the first level, e.g. a whole Attribute, the procedure described in this clause merges the provided information with the existing information up to an arbitrary depth, e.g. including going into JSON objects representing a Property value.

When merging NGSI-LD Entities 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.

With respect to merge operations, implementations shall perform an algorithm equivalent to the one described below (adapted from IETF RFC 7396 [16], in order to observe the name to URI expansion rules and JSON-LD *null* processing):

- For each member of the Fragment perform the term to URI expansion.
- If the provided Fragment (a JSON Merge Patch document) 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 contained by the target, the target member value is **merged** with the value given in the Fragment. NGSI-LD Nulls within the Fragment are given special meaning to indicate the removal of existing values within the target member value. In the case of a member representing a reified Property or Relationship including a *datasetId*, such member is only updated if the *datasetId* is the same, otherwise the member of the Fragment is added as a new instance to the target. If no *datasetId* is present, the default Attribute instance is targeted and merged if present and otherwise added. In case of a member *type* (of an Entity) in Entity Fragments, all included Entity Types are added, if they are not already contained in the *type* member of the target.

- For each member of the Fragment, whose value is an NGSI-LD Null, contained by the target, the target member is removed. In the case of deleting a specific Attribute instance with a *datasetId*, the handling shall be according to the description in clause 5.6.5. A *datasetId* cannot be deleted by setting it to the value *"urn:ngsi-ld:null"*.

EXAMPLE 1: Given an Entity containing the following Property:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 25,
    "unitCode": "CEL"
    "observedAt": "2022-03-14T01:59:26.535Z"
  }
}
```

Applying a **merge entity** operation (as defined in clause 5.6.17) onto the Entity as a whole, with the following Entity Fragment payload:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 100,
    "observedAt": "2022-03-14T13:00:00.000Z"
  }
}
```

Results in the update of the "value" and "observedAt" sub-Attributes and leaves the "unitCode" sub-Attribute untouched, as shown:

```
{
  "temperature": {
    "type" : "Property",
    "value" : 100,
    "unitCode": "CEL",
    "observedAt": "2022-03-14T13:00:00.000Z"
  }
}
```

EXAMPLE 2: Given an Entity containing the following Property:

```
{
  "address": {
    "type" : "Property",
    "value" : {
          "street": "Straße des 17. Juni",
          "city": "Berlin",
          "country": "Germany"
    }
  }
}
```

Applying a **merge entity** operation (as defined in clause 5.6.17) onto the Entity as a whole with the following Entity Fragment payload:

```
{
  "address": {
    "type" : "Property",
    "value" : {
          "street": "Pariser Platz",
          "country": "urn:ngsi-ld:null"
    }
  }
}
```

Results in the updating of the "address" Attribute value applying the JSON Merge Patch processing rules as defined in IETF RFC 7396 [16], updating "street" and removing "country" resulting as shown:

```
{
  "address": {
    "type" : "Property",
    "value": {
          "street": "Pariser Platz",
          "city": "Berlin"
    }
}
```

5.6 Context Information Provision

5.6.1 Create Entity

5.6.1.1 Description

This operation allows creating a new NGSI-LD Entity.

5.6.1.2 Use case diagram

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

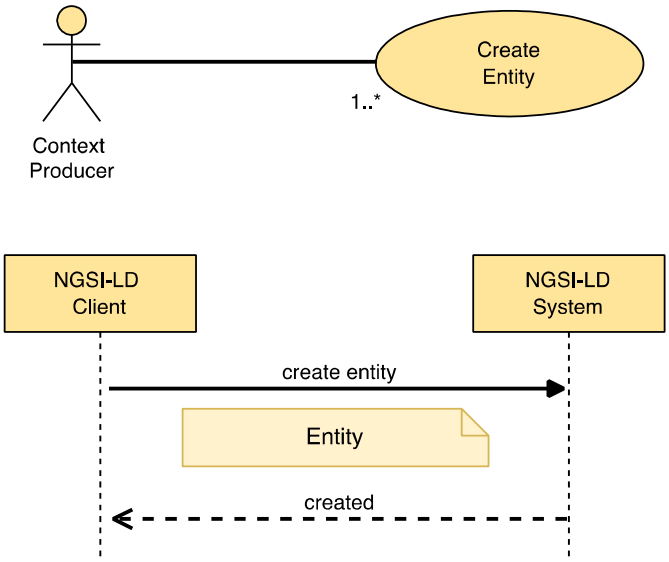

Figure 5.6.1.2-1: Create entity use case

5.6.1.3 Input data

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

5.6.1.4 Behaviour

Implementations shall exhibit the following behaviour:

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.

- If an **exclusive** Context Source Registration already exists for this id (URI), Attributes from matching input data are forwarded for remote processing:

  - For matching Registrations where the Create Entity operation is supported, the operation is forwarded to the registration endpoint. If the endpoint then raises an error, this shall result in an error in case the complete create failed or in a partial success if some parts of the create succeeded.

  - For matching Registrations where the Create Entity operation is not supported, this shall result in an error of type *Conflict* if the complete Create Entity operation failed or in a partial success if some parts of it succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.

- If any **redirect** Context Source Registrations exist that match against the input data, that input data is forwarded for remote processing by one or more matching endpoints:

  - For matching Registrations where the Create Entity operation is supported, matching input data is forwarded. If any such endpoint then raises an error, this shall result in an error in case the complete create has failed or in a partial success if some parts of the create has succeeded.

  - For matching **redirect** Registrations where the Create Entity operation is not supported, this shall result in an error of type *Conflict* if the complete Create Entity operation failed or in a partial success if some parts of it succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.

- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing by matching endpoints in case the Create Entity operation is supported.

- If the Entity already exists locally this shall result in an error of type *AlreadyExists*, if the complete Create Entity operation has failed or in a partial success if some parts of it has succeeded.

- Any remaining input data shall be used to create the Entity locally.

5.6.1.5 Output data

None.

5.6.2 Update 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 Attributes within an NGSI-LD system as shown in figure 5.6.2.2-1.

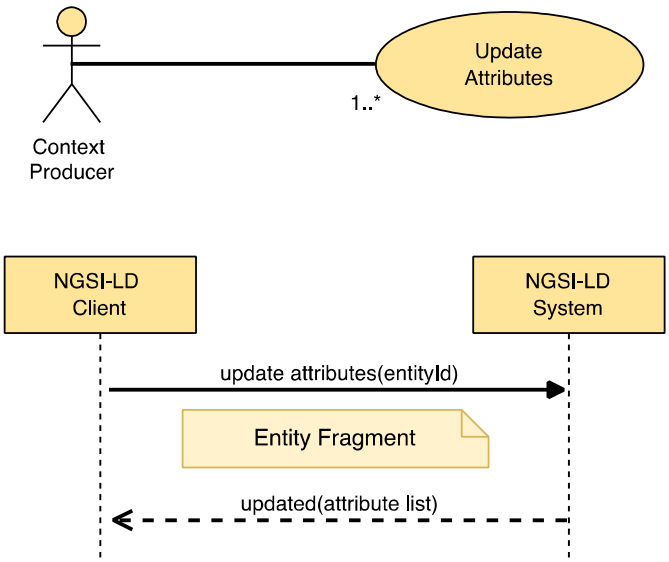

Figure 5.6.2.2-1: Update Attributes use case

5.6.2.3 Input data

- A URI representing the id of the Entity to be updated (target Entity).
- A selector of Entity types as specified by clause 4.17 (optional).
- A JSON-LD document representing an NGSI-LD Entity Fragment.

5.6.2.4 Behaviour

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent to the target entity held locally, and no matching registrations apply, an error of type *ResourceNotFound* shall be raised.
- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation. NGSI-LD Nulls should be supported by this operation. If NGSI-LD Nulls are found in the payload, but are not supported, an error of type *OperationNotSupported* shall be raised.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, Attributes from matching input data are forwarded for remote processing. For each matching registration:
  - If the Update Attributes operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Update Attributes operation is not supported by the matched registration, this shall result in an error of type *Conflict* if the complete update failed or in a partial success if some parts of the update succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.
- If there are remaining Attributes, for any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing to matching endpoints in case the Update Attributes operation is supported by the matched registration.
- Then, implementations shall perform a partial update patch operation over the remains of the target Entity as mandated by clause 5.5.8, using the following procedure:

- For each Attribute (Property or Relationship) included by the Entity Fragment at root level:
  - If the target Entity does not include a matching Attribute (considering term expansion rules as mandated by clause 5.5.7) then such Attribute shall be appended to the target Entity.
  - If the target Entity already includes a matching Attribute (considering term expansion rules as mandated by clause 5.5.7):
    - If a *datasetId* is present in the Attribute included by the Entity Fragment:
      - If an Attribute instance in the target Entity has the same *datasetId* and the Attribute value is not NGSI-LD Null, then the existing Attribute instance with the specified *datasetId* in the target Entity shall be replaced by the new one supplied.
      - If an Attribute instance in the target Entity has the same *datasetId* and the Attribute value is NGSI-LD Null then the existing Attribute instance with the specified *datasetId* in the target Entity shall be deleted.
      - Otherwise the Attribute instance with the specified *datasetId* shall be appended to the target Entity.
    - If no *datasetId* is present in the Attribute included by the Entity Fragment, the default Attribute instance is targeted:
      - If the default Attribute instance is present and the Attribute value is not NGSI-LD Null, then the existing Attribute in the target Entity shall be replaced by the new one supplied.
      - If the default Attribute instance is present and the Attribute value is NGSI-LD Null, then the existing Attribute in the target Entity shall be deleted.
      - Otherwise the default Attribute instance shall be appended to the target Entity.
- If *type* is included in the Fragment and it includes Entity Type Names that are not yet in the target Entity, add them to the list of Entity Type Names of the target Entity.
- If *scope* is included in the Fragment and the target entity includes *scope*, replace the scope by the one included in 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 Attributes

5.6.3.1 Description

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

5.6.3.2 Use case diagram

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

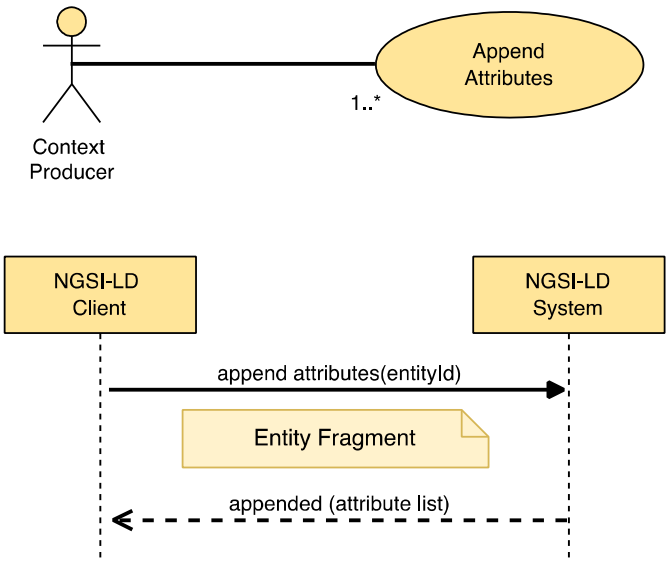

Figure 5.6.3.2-1: Append Attributes use case

5.6.3.3 Input data

- A URI representing the id of the E to be modified (target Entity).
- A selector of Entity types as specified by clause 4.17 (optional).
- A JSON-LD document representing an NGSI-LD Entity Fragment.
- An optional flag indicating whether overwriting existing Attributes within the append operation should be permitted or denied. By default, Attribute overwrites are permitted.

5.6.3.4 Behaviour

The following behaviour shall be exhibited by compliant implementations:

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about this Entity, because there is no existing Entity which id (URI), and where specified type, is equivalent held locally to the one passed as parameter, and no matching registrations apply, an error of type *ResourceNotFound* shall be raised.
- The behaviour defined in clause 5.5.4 on JSON-LD validation.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the Attributes from matching input data are forwarded for remote processing. For each matching registration:
  - If the Append Attributes operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Append Attributes operation is not supported by the matched registration, this shall result in an error of type *Conflict* if the complete append failed or in a partial success if some parts of the append succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.
- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing to matching endpoints in case the Append Attributes operation is supported.

- Then, implementations shall perform an Append Attributes operation over the remains of the target Entity as using the following procedure.
- For each Attribute (Property or Relationship) included by the Entity Fragment at root level:
  - If a *datasetId* is present in the Attribute included by the Entity Fragment:
    - If no Attribute instance of the same target Entity exists that has the same *datasetId*, then such an Attribute shall be appended to the target Entity.
    - If an Attribute instance of the same target Entity exists that has the same *datasetId:*
      - If overwrite is allowed, then the existing Attribute with the specified *datasetId* in the target Entity shall be replaced by the new one supplied.
      - If overwrite is not allowed, the existing Attribute with the specified *datasetId* in the target Entity shall be left untouched.
  - If no *datasetId* is present in the Attribute included by the Entity Fragment:
    - If no default Attribute instance of the same target Entity exists, then such Attribute shall be appended to the target Entity.
    - If a default Attribute instance of the same target Entity exists:
      - If overwrite is allowed, then the existing default Attribute in the target Entity shall be replaced by the new one supplied.
      - If overwrite is not allowed the existing default Attribute in the target Entity shall be left untouched.
- If *type* is included in the Fragment and it includes Entity Type Names that are not yet in the target Entity, add them to the list of Entity Type Names of the target Entity.
- If *scope* is included in the Fragment and overwrite is allowed, the scope of the target Entity will become the one included in the Fragment. Otherwise, the Scopes in the Fragment that are not part of the value of *scope* of the target Entity will be appended to the value of the *scope* of the target Entity. If there is more than one Scope, the value of *scope* is represented as a JSON array containing all Scopes.

5.6.3.5 Output data

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

5.6.4 Partial Attribute update

5.6.4.1 Description

This operation allows performing a **partial update on an NGSI-LD Entity's Attribute** (Property or Relationship). A partial update only changes the elements provided in an Entity Fragment, leaving the rest as they are. This operation supports the deletion of sub-Attributes but not the deletion of the whole Attribute itself.

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.

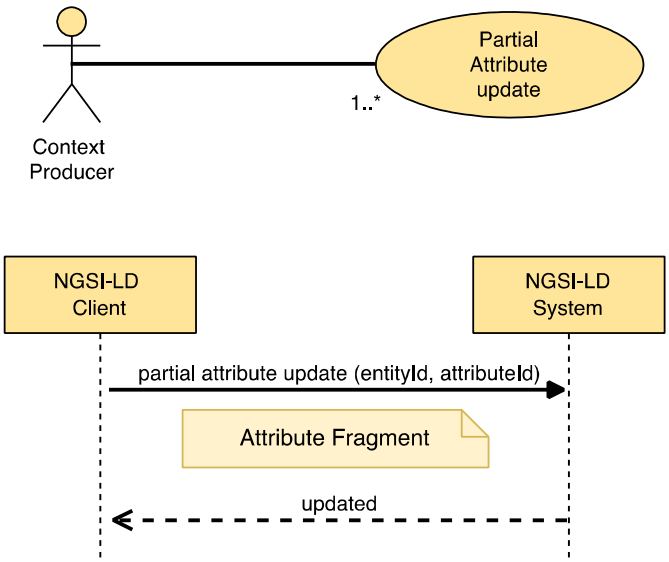

Figure 5.6.4.2-1: Partial Attribute update use case

5.6.4.3 Input data

- Entity Id (URI) of the concerned Entity, the target Entity.
- A selector of Entity types as specified by clause 4.17 (optional).
- Target Attribute (Property or Relationship) to be modified, identified by a name.
- A JSON-LD document representing an NGSI-LD Attribute 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.
- The behaviour defined in clause 5.5.4 on JSON-LD validation. NGSI-LD Nulls should be supported by this operation. If NGSI-LD Nulls are found in the payload, but are not supported, an error of type *OperationNotSupported* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally, and no matching registrations apply, then an error of type ResourceNotFound shall be raised.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the Attributes from matching input data are forwarded for remote processing. For each matching registration:

  - If the Partial Attribute update operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Partial Attribute update operation is not supported by the matched registration an, this shall result in an error of type *Conflict* in case the complete partial Attribute update failed, or in a partial success if some parts of the partial Attribute update succeeded.

  No further processing is required.
- For any **inclusive** Context Source Registrations that exist and match against the input data, that input data is also forwarded for remote processing to matching endpoints.

- 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 Attribute is *scope*, replace *scope* in the target Entity.
- If the target Entity does not contain the target Attribute:
  - as a default instance in case no *datasetId* is present;
  - as an instance with the specified *datasetId* if present;

  then an error of type *ResourceNotFound* shall be raised.
- Perform a partial update patch operation on the target Attribute following the algorithm mandated by clause 5.5.8. If present in the provided NGSI-LD Entity Fragment, the type of the Attribute has to be the same as the type of the targeted Attribute fragment, i.e. it is not allowed to change the type of an Attribute.

5.6.4.5 Output data

None.

5.6.5 Delete Attribute

5.6.5.1 Description

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

5.6.5.2 Use case diagram

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

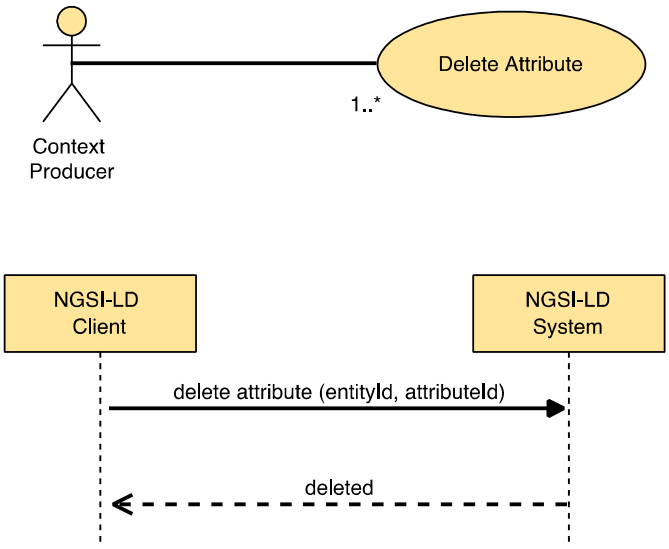

Figure 5.6.5.2-1: Delete Attribute use case

5.6.5.3 Input data

- Entity id (URI) of the concerned Entity, the target Entity.
- A selector of Entity types as specified by clause 4.17 (optional).
- Target Attribute (Property or Relationship) to be deleted, identified by a Name.

- An optional parameter identifying the *datasetId* of the target Attribute instance to be deleted. Otherwise the default Attribute instance is targeted.
- An optional flag "deleteAll" indicating whether also all target Attribute instances with a *datasetId* are to be deleted.
- An optional JSON-LD @context.

5.6.5.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name or it is not present, then an error of type *BadRequestData* shall be raised.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the input data is forwarded. For each matching registration:
  - If the Delete Attribute operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Delete Attribute update operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete delete Attribute failed, or in a partial success if some parts of the delete Attribute succeeded.

  No further processing is required.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally, and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- For any **inclusive** Context Source Registrations that exist and match against the input data, that input data is also forwarded for remote processing to matching endpoints.
- Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If the target Attribute is *scope*, remove the *scope* Attribute from the target Entity.
- If the *deleteAll* flag is set, remove all target Attribute instances from the target Entity.
- Otherwise:
  - if a *datasetId* parameter is provided, remove only the target Attribute instance from the given dataset whose *datasetId* matches the parameter;
  - if no *datasetId* parameter is provided, remove the default target Attribute instance 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 an NGSI-LD Entity.

5.6.6.2 Use case diagram

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

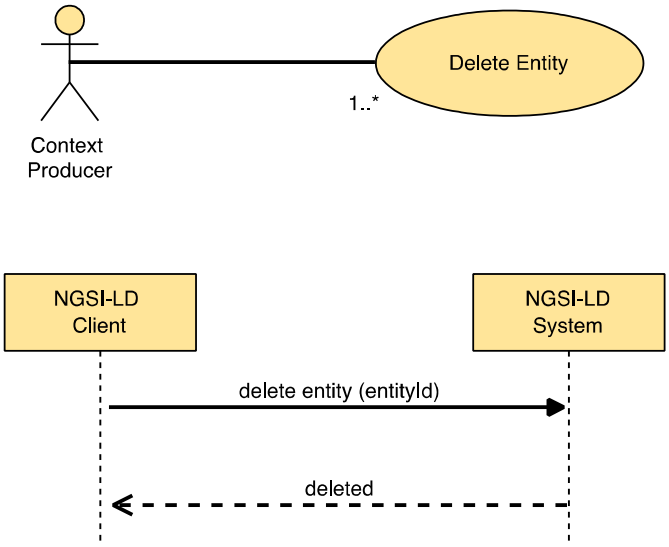

Figure 5.6.6.2-1: Delete Entity use case

5.6.6.3 Input data

- Entity Id (URI) of the Entity to be deleted, the target Entity.
- A selector of Entity types as specified by clause 4.17 (optional).

5.6.6.4 Behaviour

- If the target Entity id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally, and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- If an **exclusive** or Context Source Registration matches against the id, the request is forwarded for remote processing. For each matching registration:
  - If the Delete Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Delete Entity update operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete delete Entity failed, or in a partial success if some parts of the delete Entity succeeded.
- If any **redirect** Context Source Registrations exist that match against the input data, that input data is forwarded for remote processing by one or more matching endpoints:
  - For matching Registrations where the Create Entity operation is supported, matching input data is forwarded. If any such endpoint then raises an error, the implementation shall return with the error(s) raised.
  - For matching **redirect** Registrations where the Create Entity operation is not supported, this shall result in an error of type *Conflict* in case the complete delete Entity failed, or in a partial success if some parts of the delete Entity succeeded.
- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing by matching endpoints.

- The input data shall be used to remove the entity locally if it exists.

5.6.6.5 Output data

None.

5.6.7 Batch Entity Creation

5.6.7.1 Description

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

5.6.7.2 Use case diagram

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

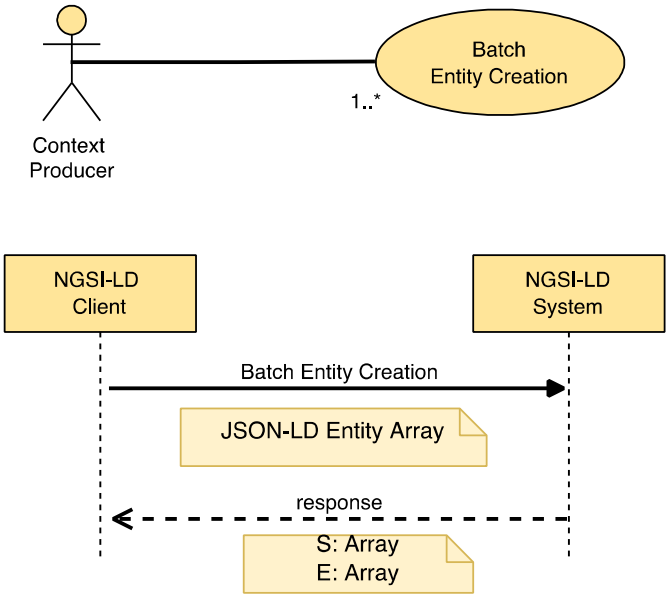

Figure 5.6.7.2-1: Create a batch of Entities use case

5.6.7.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an NGSI-LD Entity as mandated by clause 5.2.4. See clause 5.5.11.1 for information on behaviour when there is more than one instance of the same entity in the input Array.

5.6.7.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items an error of type *BadRequestData* shall be raised.
- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity successfully created. S shall be initialized to the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized to the empty array.

- For each Context Source Registration CSR in the Context Registry:
  - Let IN be a copy of the original input array.
  - Remove from IN all Entities not matched by CSR and remove non-matching Attributes from the remaining Entities.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **exclusive** Context Source Registration, which is not CSR itself.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **redirect** Context Source Registration, unless CSR is a redirect Context Source Registration itself.
  - If IN is empty, continue with the next Context Source Registration if there is any.
  - If the Batch Entity Creation operation is supported by CSR:
    - Forward the Batch Entity Creation request with IN as input Array.
    - Merge the returned list of Entities successfully created with S.
    - Merge the returned list of Entities in Error with E.
  - Otherwise, if the Create Entity operation (clause 5.6.1) is supported by CSR:
    - For each Entity EN in the input array:
      - Forward a Create Entity request for Entity EN.
      - Merge any successful result(s) for Entity EN created with S.
      - Merge any error result(s) for Entity EN created with E.
  - Otherwise:
    - In case CSR is an **exclusive** or **redirect** Context Source Registration, add an Error of type *Conflict* for each Entity in IN to E.
- For each of the NGSI-LD Entities included in the input Array execute the behaviour defined by clause 5.6.1, but limited to a local operation, as follows:
  - If the Entity was successfully created, then add the corresponding Entity Id to the S array.
  - If the Entity creation failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the related *ProblemDetails*.

5.6.7.5 Output data

- the list of Entities successfully created (S Array), if all Entities were created correctly; or
- the list of Entities successfully created (S Array) and the list of Entities in error (E Array), if only some or none of the Entities were created.

5.6.8 Batch Entity Creation or Update (Upsert)

5.6.8.1 Description

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

5.6.8.2 Use case diagram

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

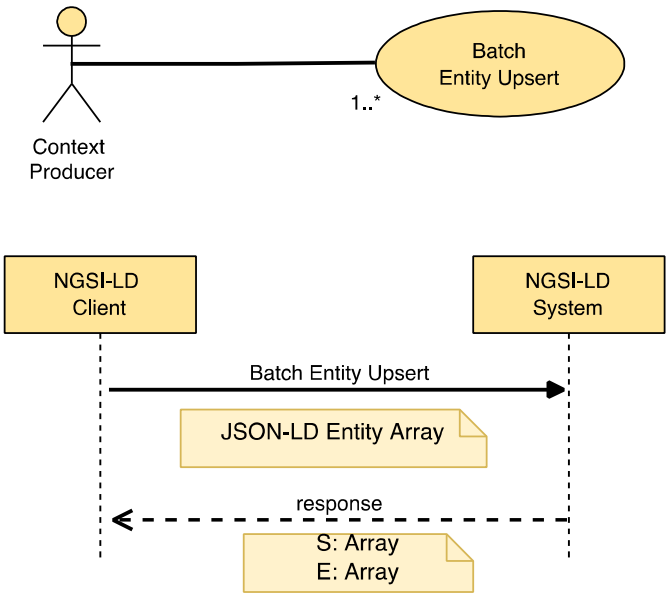

Figure 5.6.8.2-1: Upsert a batch of Entities use case

5.6.8.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an Entity as mandated by clause 5.2.4. See clause 5.5.11.2 for information on behaviour when there is more than one instance of the same entity in the input Array.
- An optional flag indicating the update mode (only applies in case the Entity already exists):
  - Replace. All the existing Entity content shall be replaced (default mode).
  - Update. Existing Entity content shall be updated.

5.6.8.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items, an error of type *BadRequestData* shall be raised.
- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized to the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized to the empty array.
- For each Context Source Registration CSR in the Context Registry:
  - Let IN be a copy of the original input array.
  - Remove from IN all Entities not matched by CSR and remove non-matching Attributes from the remaining Entities.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **exclusive** Context Source Registration, which is not CSR itself.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **redirect** Context Source Registration, unless CSR is a redirect Context Source Registration itself.
  - If IN is empty, continue with the next Context Source Registration if there is any.

- If the Batch Entity Creation or Update (Upsert) operation is supported by CSR:
  - Forward the Batch Entity Creation or Update (Upsert) request with IN as input Array.
  - Merge the returned list of Entities successfully created with S.
  - Merge the returned list of Entities in Error with E.
- Otherwise, if the Create Entity operation (clause 5.6.1) is supported by CSR:
  - For each Entity EN in the input array:
    - Forward a Create Entity request for Entity EN.
    - If an error of type *AlreadyExists* is returned:
      - If the Replace Entity operation (clause 5.6.18) is supported by CSR and the value of the update mode flag is *Replace* or the flag is not set, forward a Replace Entity request for Entity EN.
      - Otherwise, if the Update Attributes operation (clause 5.6.2) is supported by CSR and the value of the update mode flag is *Update*, forward an Update Attributes request for Entity EN.
      - Otherwise add an *OperationNotSupported* Error for Entity EN related to CSR to E.
    - Merge any successful result(s) for Entity EN created or updated with S.
    - Merge any error result(s) for Entity EN created or updated with E.
- Otherwise, if the Replace Entity operation (clause 5.6.18) is supported by CSR and the value of the update mode flag is *Replace* or the flag is not set:
  - Forward a Replace Entity request for Entity EN.
  - Merge any successful result(s) for Entity EN updated with S.
  - Merge any error result(s) for Entity EN updated with E.
- Otherwise, if the Update Attributes operation (clause 5.6.2) is supported by CSR and the value of the update mode flag is *Update*:
  - Forward an Update Attributes request for Entity EN.
  - Merge any successful result(s) for Entity EN updated with S.
  - Merge any error result(s) for Entity EN updated with E.
- Otherwise:
  - In case CSR is an **exclusive** or **redirect** Context Source Registration, add an Error of type *Conflict* for each Entity in IN to E.

- For each of the NGSI-LD Entities included in the input Array implementations shall:
  - Create the Entity locally if it does not exist (i.e. no Entity with the same Entity Id is present) executing the behaviour defined by clause 5.6.1, but limited to a local operation.
  - If there were an existing Entity with the same Entity Id, it shall be completely replaced by the new Entity content provided, if the requested update mode is 'replace'.
  - If there were an existing Entity with the same Entity Id, the behaviour defined by clause 5.6.3 shall be executed, but limited to a local operation, if the requested update mode is 'update'.
- If successful, the local creation or update shall be added to S. If while processing an Entity there is any kind of error or abnormal situation, a *BatchEntityError* shall be added to E containing the failed Entity Id and the related *ProblemDetails*.

5.6.8.5 Output data

- none (if all Entities already existed and are successfully updated); or
- the list of Entities successfully created (S Array), if all Entities not existing prior to this request have been successfully created and the others have been successfully updated; or
- the list of Entities successfully created or updated (S Array), and the list of Entities in error (E Array), if only some or none of the Entities have been successfully created or updated.

5.6.9 Batch Entity Update

5.6.9.1 Description

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

5.6.9.2 Use case diagram

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

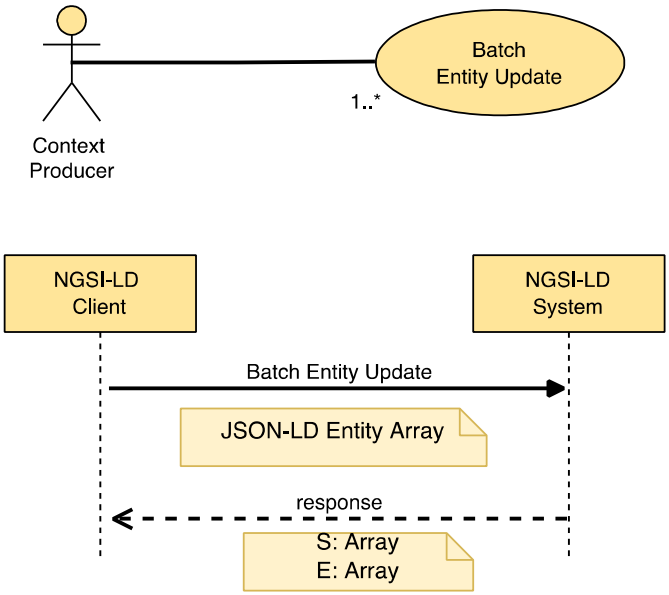

Figure 5.6.9.2-1: Update a batch of Entities use case

5.6.9.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an Entity as mandated by clause 5.2.4. See clause 5.5.11.3 for information on behaviour when there is more than one instance of the same entity in the input Array.
- An optional flag indicating whether Attributes shall be overwritten or not. By default, Attributes will be overwritten.

5.6.9.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items, an error of type *BadRequestData* shall be raised.
- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.

- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized as the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized as the empty array.
- For each Context Source Registration CSR in the Context Registry:
  - Let IN be a copy of the original input array.
  - Remove from IN all Entities not matched by CSR and remove non-matching Attributes from the remaining Entities.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **exclusive** Context Source Registration, which is not CSR itself.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **redirect** Context Source Registration, unless CSR is a redirect Context Source Registration itself.
  - Remove all Entities without Attributes from IN.
  - If IN is empty, continue with the next Context Source Registration if there is any.
  - If the Batch Entity Update operation is supported by CSR:
    - Forward the Batch Entity Update request with IN as input Array.
    - Merge the returned list of Entities successfully created with S.
    - Merge the returned list of Entities in Error with E.
  - Otherwise, if the Update Attributes operation (clause 5.6.2) is supported by CSR and Attribute overwrite is permitted:
    - For each Entity EN in the input array:
      - Forward an Update Attributes request for Entity EN.
      - Merge any successful result(s) for Entity EN updated with S.
      - Merge any error result(s) for Entity EN updated with E.
  - Otherwise, if the Append Attributes operation (clause 5.6.3) is supported by CSR and Attribute overwrite is not permitted:
    - For each Entity EN in the input array:
      - Forward an Append Attributes request for Entity EN with Attribute overwrite disabled.
      - Merge any successful result(s) for Entity EN updated with S.
      - Merge any error result(s) for Entity EN updated with E.
  - Otherwise:
    - In case CSR is an **exclusive** or **redirect** Context Source Registration, add an Error of type *Conflict* for each Entity in IN to E.
- For each of the NGSI-LD Entities included in the input Array execute the behaviour defined by clause 5.6.3, but limited to a local operation, as follows:
  - If the Entity was successfully updated (Attributes appended), then add the corresponding Entity Id to the S array.
  - If the Entity update failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the *ProblemDetails* associated.

5.6.9.5 Output data

- none (if all Entities are successfully updated); or
- the list of Entities successfully updated (S Array), and the list of Entities in error (E Array), if only some or none of the Entities have been successfully updated.

5.6.10 Batch Entity Delete

5.6.10.1 Description

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

5.6.10.2 Use case diagram

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

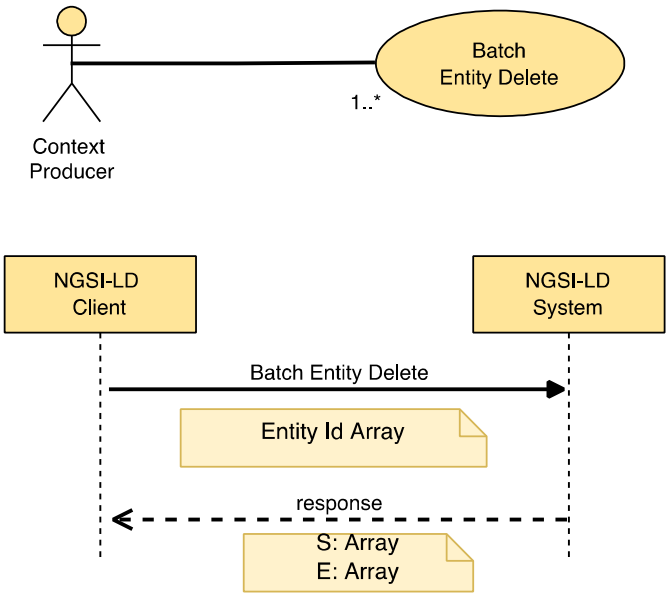

Figure 5.6.10.2-1: Delete a batch of Entities use case

5.6.10.3 Input data

- A JSON-LD Array containing a list of Entity Ids (URIs) that are requested to be deleted. See clause 5.5.11.4 for information on behaviour when there is more than one instance of the same Entity Id in the input Array.

5.6.10.4 Behaviour

Implementations shall exhibit the following behaviour:

- If the input Array is empty or contains a *null* value in any of its items, an error of type *BadRequestData* shall be raised.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized to the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized to the empty array.
- For each Context Source Registration CSR in the Context Registry:
  - Let IN be a copy of the original input array.

- Remove from IN all Entities not matched by CSR and remove non-matching Attributes from the remaining Entities.
- Remove all Attributes from the remaining Entities in IN for which there is a matching **exclusive** Context Source Registration, which is not CSR itself.
- Remove all Attributes from the remaining Entities in IN for which there is a matching **redirect** Context Source Registration, unless CSR is a redirect Context Source Registration itself.
- Remove all Entities without Attributes from IN.
- If IN is empty, continue with the next Context Source Registration if there is any.
- If the Batch Entity Delete operation is supported by CSR:
  - Forward the Batch Entity Delete request with IN as input Array.
  - Merge the returned list of Entities successfully created with S.
  - Merge the returned list of Entities in Error with E.
- Otherwise, if the Delete Entity operation (clause 5.6.6) is supported by CSR:
  - For each Entity EN in the input array:
    - Forward a Delete Entity request for Entity EN.
    - Merge any successful result(s) for Entity EN deleted with S.
    - Merge any error result(s) for Entity EN deleted with E.
- Otherwise:
  - In case CSR is an **exclusive** or **redirect** Context Source Registration, add an Error of type *Conflict* for each Entity in IN to E.

- For each of the NGSI-LD Entity Ids included in the input Array execute the behaviour defined by clause 5.6.6, but limited to a local operation, as follows:
  - If the Entity corresponding to an Entity Id was successfully deleted, then add such Entity Id to the S array.
  - If the Entity deletion failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the related *ProblemDetails*.

5.6.10.5 Output data

- none (if all Entities that already existed are successfully deleted); or
- the list of Entities successfully deleted (S Array), and the list of Entities in error (E Array), if some or all of the Entities have not been successfully deleted.

5.6.11 Create or Update (Upsert) Temporal Representation of an Entity

5.6.11.1 Description

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

5.6.11.2 Use case diagram

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

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

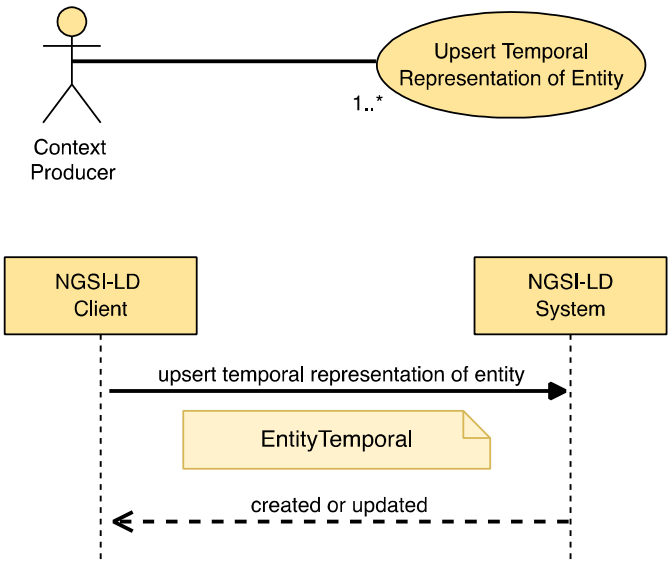

Figure 5.6.11.2-1: Create or Update (Upsert) Temporal Representation of Entity use case

5.6.11.3 Input data

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

5.6.11.4 Behaviour

Implementations shall exhibit the following behaviour:

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- If an **exclusive** Context Source Registration already exists for this id (URI), Attributes from matching input data are forwarded for remote processing:
  - For matching Registrations where the Create or Update (Upsert) Temporal Representation of Entity operation is supported, the operation is forwarded to the registration endpoint. If the endpoint then raises an error, this shall result in an error in case the complete Create or Update (Upsert) Temporal Representation of Entity operation failed or in a partial success if some parts of it succeeded.
  - For matching Registrations where the Create Entity operation is not supported, this shall result in an error of type *Conflict* in case the complete Create or Update (Upsert) Temporal Representation of Entity operation failed or in a partial success if some parts of it succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.
- If any **redirect** Context Source Registrations exist that match against the input data, that input data is forwarded for remote processing by one or more matching endpoints:
  - For matching Registrations where the Create or Update (Upsert) Temporal Representation of Entity operation is supported, matching input data is forwarded. If any such endpoint then raises an error, this shall result in an error in case the complete Create or Update (Upsert) Temporal Representation of Entity operation failed or in a partial success if some parts of it succeeded.
  - For matching **redirect** Registrations where the Create or Update (Upsert) Temporal Representation of Entity operation is not supported, this shall result in an error of type *Conflict* in case the complete Create or Update (Upsert) Temporal Representation of Entity operation failed or in a partial success if some parts of it succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.
- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing by matching endpoints.

- If the NGSI-LD endpoint already knows about this Temporal Representation of an Entity, because there is an existing Temporal Representation of an Entity whose id (URI) is the same, then all the Attribute instances included by the Temporal Representation shall be added to the existing Entity as mandated by clause 5.6.12. If *type* is included in the EntityTemporal Fragment and it includes Entity Type Names that are not yet in the target Temporal Representation of an Entity, add them to the list of Entity Type Names of the target Temporal Representation of an Entity.
- Otherwise, implementations shall create the provided Temporal Representation of an Entity.

5.6.11.5 Output data

None.

5.6.12 Add Attributes to Temporal Representation of an Entity

5.6.12.1 Description

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

5.6.12.2 Use case diagram

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

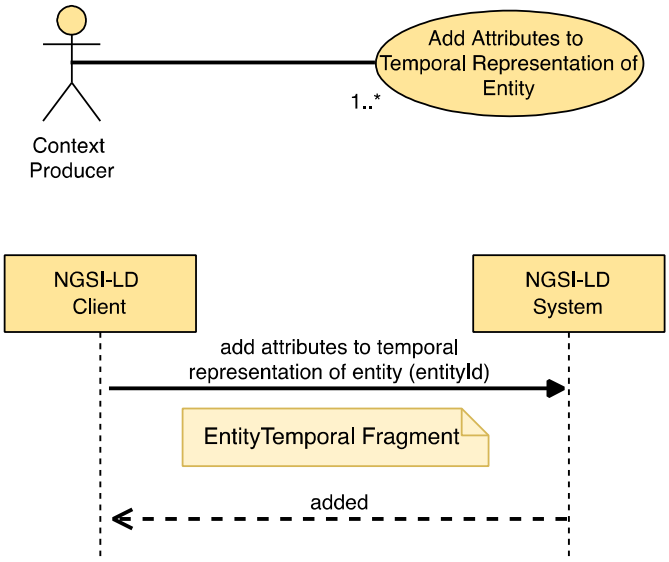

Figure 5.6.12.2-1: Add Attributes to Temporal Representation of Entity use case

5.6.12.3 Input data

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

5.6.12.4 Behaviour

The following behaviour shall be exhibited by compliant implementations:

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the Temporal Representation of the target Entity, because there is no existing Temporal Representation of an Entity whose id (URI) is equivalent to the one passed as parameter held locally and no matching registrations apply, an error of type *ResourceNotFound* shall be raised.
- The behaviour defined in clause 5.5.4 on JSON-LD validation.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the Attributes from matching input data are forwarded for remote processing. For each matching registration:
  - If the Add Attributes to Temporal Representation of an Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Add Attributes to Temporal Representation of an Entity operation is not supported by the matched registration, this shall result in an error of type *Conflict* if the complete Add Attributes to Temporal Representation of an Entity operation failed or in a partial success if some parts of it succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.
- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing to matching endpoints.
- If the target Entity exists locally and matches against the remaining input data, implementations shall do the following:
  - For each Attribute (Property or Relationship) instance included by the *EntityTemporal* Fragment at root level:
    - The Attribute (considering term expansion rules as mandated by clause 5.5.7) instance(s) shall be added to the target Entity.
- If *type* is included in the EntityTemporal Fragment and it includes Entity Type Names that are not yet in the target Temporal Representation of an Entity, add them to the list of Entity Type Names of the target Temporal Representation of an Entity.

5.6.12.5 Output data

None.

5.6.13 Delete Attribute from Temporal Representation of an Entity

5.6.13.1 Description

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

5.6.13.2 Use case diagram

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

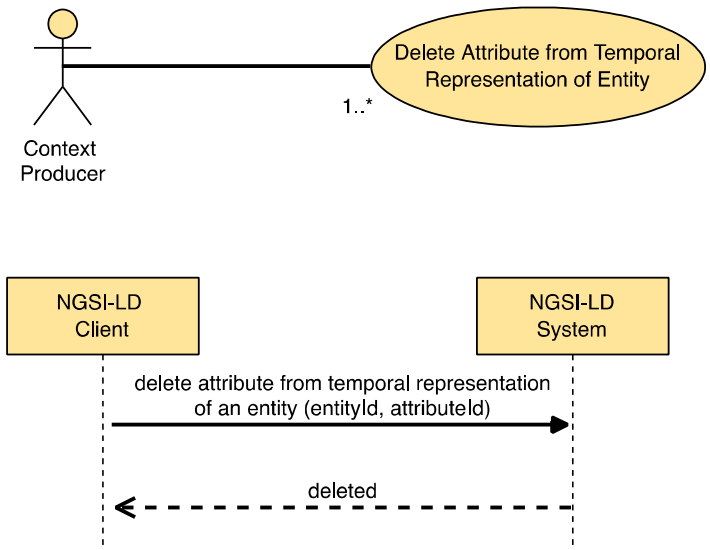

Figure 5.6.13.2-1: Delete Attribute from Temporal Representation of Entity use case

5.6.13.3 Input data

- Entity id (URI) of the target Entity which Temporal Representation is to be modified.
- Target Attribute (Property or Relationship) to be deleted, identified by a Name.
- An optional parameter identifying the dataset (*datasetId*) of the target Attribute instance to be deleted.
- An optional parameter, a flag, (*deleteAll*) indicating whether all target Attribute instances are to be deleted, regardless of *datasetId*.
- An optional JSON-LD @context.

5.6.13.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Temporal Representation of an Entity whose id (URI) is equivalent held locally and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the input data is forwarded. For each matching registration:
  - If the Delete Attribute from Temporal Representation of an Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Delete Attribute from Temporal Representation of an Entity operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete Delete Attribute from Temporal Representation of an Entity failed, or in a partial success if some parts of it succeeded.

  No further processing is required.
- For any **inclusive** Context Source Registrations that exist and match against the input data, that input data is also forwarded for remote processing to matching endpoints.

- If the target Entity exists locally, implementations shall do the following:
  - Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If the *deleteAll* flag is set, remove all target Attribute instances from the target Entity.
- Otherwise:
  - if a *datasetId* parameter is provided, remove only any target Attribute instance from the given dataset;
  - if no *datasetId* parameter is provided, remove only the default target Attribute instance *datasetId* from the target Entity.

5.6.13.5 Output data

None.

5.6.14 Modify Attribute instance in Temporal Representation of an Entity

5.6.14.1 Description

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

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

5.6.14.2 Use case diagram

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

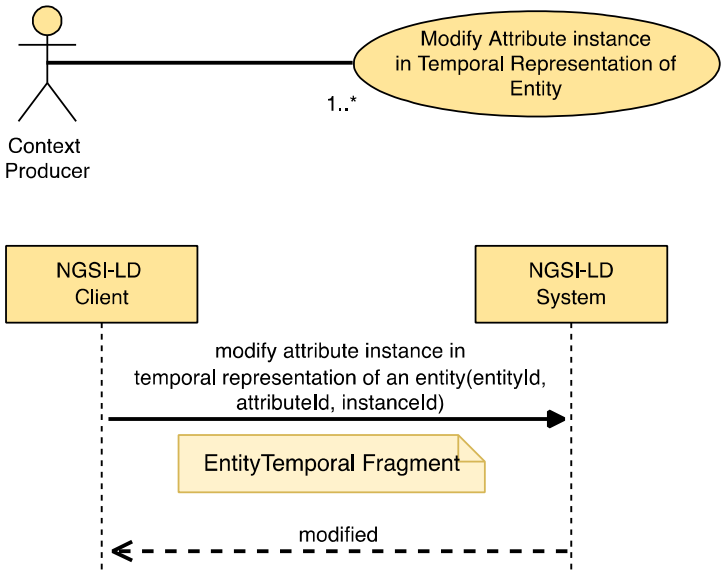

Figure 5.6.14.2-1: Modify Attribute Instance in Temporal Representation of Entity use case

5.6.14.3 Input data

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

5.6.14.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target *instanceId* is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent held locally and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the input data is forwarded. For each matching registration:
  - If the Modify Attribute instance in Temporal Representation of an Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Modify Attribute instance in Temporal Representation of an Entity operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete Modify Attribute instance in Temporal Representation of an Entity operation failed, or in a partial success if some parts of it succeeded.

  No further processing is required.
- For any **inclusive** Context Source Registrations that exist and match against the input data, that input data is also forwarded for remote processing to matching endpoints.
- If the target Entity exists locally, implementations shall do the following:
  - Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.
- If the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.
- If for the target Attribute no instance with the specified *instanceId* exists, an error of type *ResourceNotFound* shall be raised.
- Replace the target Attribute instance identified by the *instanceId* with the Attribute instance in the *EntityTemporal* Fragment. The *createdAt* property of the concerned instance shall remain unchanged, but the *modifiedAt* property shall be set to the timestamp corresponding to this modification.

5.6.14.5 Output data

None.

5.6.15 Delete Attribute instance from Temporal Representation of an Entity

5.6.15.1 Description

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

5.6.15.2 Use case diagram

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

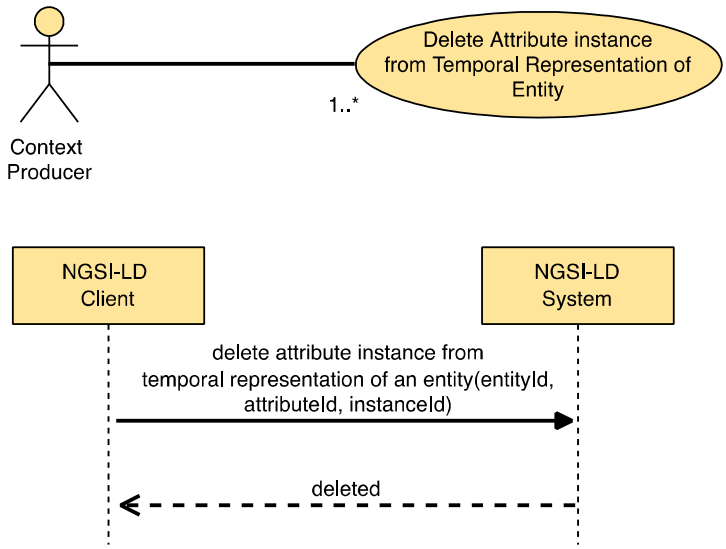

Figure 5.6.15.2-1: Delete Attribute Instance from Temporal Representation of Entity use case

5.6.15.3 Input data

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

5.6.15.4 Behaviour

- If the target Entity id is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target Attribute name is not a valid Name or it is not present, then an error of type *BadRequestData* shall be raised.
- If the target *instanceId* is not a valid URI or it is not present, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent held locally and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.

- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the input data is forwarded. For each matching registration:

  - If the Delete Attribute instance from Temporal Representation of an Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.

  - If the Delete Attribute instance from Temporal Representation of an Entity operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete Delete Attribute instance from Temporal Representation of an Entity failed, or in a partial success if some parts of it succeeded.

  No further processing is required.

- For any **inclusive** Context Source Registrations that exist and match against the input data, that input data is also forwarded for remote processing to matching endpoints.

- If the target Entity exists locally, implementations shall do the following:

  - Apply term expansion as mandated by clause 5.5.7 so that the fully qualified name (URI) associated to the target Attribute is properly obtained.

- If the Temporal Representation of the target Entity does not contain the target Attribute then an error of type *ResourceNotFound* shall be raised.

- If for the target Attribute no instance with the specified *instanceId* exists, an error of type *ResourceNotFound* shall be raised.

- Remove the instance, with the specified *instanceId*, of the target Attribute from the target Entity.

5.6.15.5 Output data

None.

5.6.16 Delete Temporal Representation of an Entity

5.6.16.1 Description

This operation allows deleting the Temporal Representation of an Entity.

5.6.16.2 Use case diagram

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

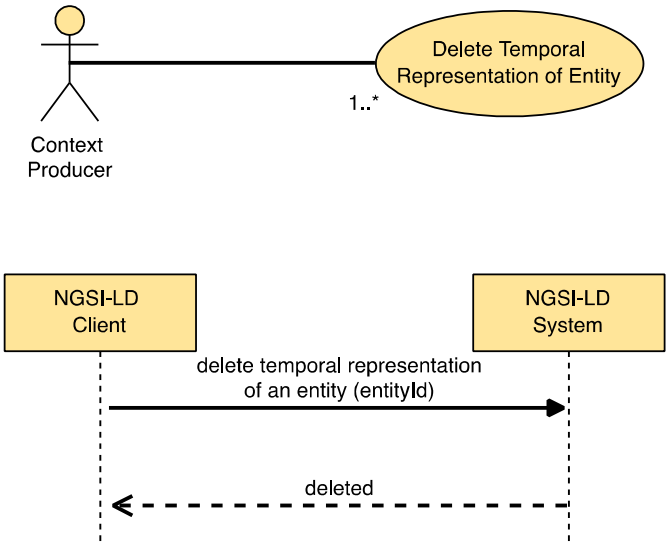

Figure 5.6.16.2-1: Delete Temporal Representation of Entity use case

5.6.16.3 Input data

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

5.6.16.4 Behaviour

- If the target Entity id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity because there is no existing Entity whose id (URI) is equivalent held locally and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the input data is forwarded. For each matching registration:
  - If the Delete Temporal Representation of Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Delete Temporal Representation of Entity operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete Delete Temporal Representation of Entity failed, or in a partial success if some parts of it succeeded.

  No further processing is required.
- For any **inclusive** Context Source Registrations that exist and match against the input data, that input data is also forwarded for remote processing to matching endpoints.
- If the target Entity exists locally, the entire Temporal Representation of the Entity shall be removed.

5.6.16.5 Output data

None.

5.6.17 Merge Entity

5.6.17.1 Description

This operation allows modification of an existing NGSI-LD Entity aligning to the JSON Merge Patch processing rules defined in IETF RFC 7396 [16] by adding new Attributes (Properties or Relationships) or modifying or deleting existing Attributes associated with an existing Entity.

5.6.17.2 Use case diagram

A Context Producer can perform a merge on an Entity within an NGSI-LD system as shown in figure 5.6.17.2-1.

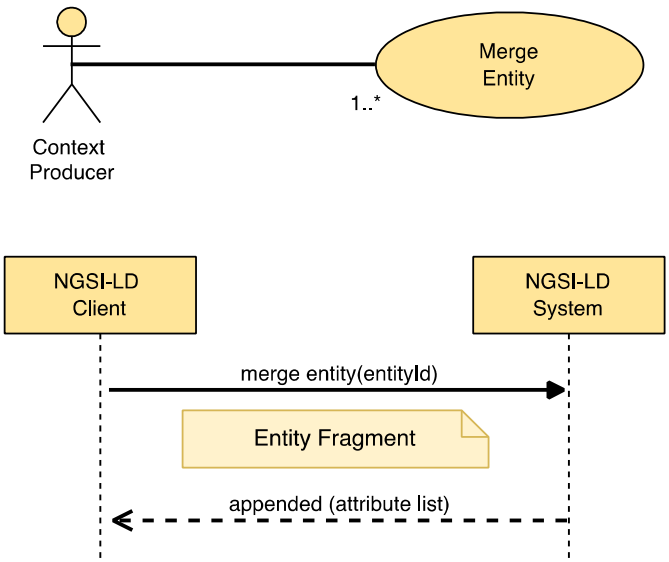

Figure 5.6.17.2-1: Merge Entity use case

5.6.17.3 Input data

- A URI representing the id of the Entity to be merged (target Entity).
- A selector of Entity types as specified by clause 4.17 (optional).
- A JSON-LD document representing an NGSI-LD Entity Fragment.
- An optional flag indicating whether the JSON-LD document contains a simplified representation of the entity.
- An optional parameter indicating a common "observedAt" timestamp to use across merged Attributes.
- An optional parameter representing a common IETF RFC 5646 [28] language tag to use across merged LanguageMap Attributes.

5.6.17.4 Behaviour

The following behaviour shall be exhibited by compliant implementations:

- If the Entity Id is not present or it is not a valid URI then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally, and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.

- If an **exclusive** or **redirect** Context Source Registration matches against the input data, Attributes from matching input data are forwarded. For each matching registration:

  - If the Merge Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.

  - If the Merge Entity operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete Merge Entity operation failed, or in a partial success if some parts of it succeeded.

  The matching Attributes are then removed from the Fragment and not processed further.

- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing to matching endpoints.

- The behaviour defined in clause 5.5.4 on JSON-LD validation. NGSI-LD Nulls should be supported by this operation. If NGSI-LD Nulls are found in the payload, but are not supported, an error of type *OperationNotSupported* shall be raised.

  Then, implementations shall perform a merge operation over the target Entity as mandated by clause 5.5.12, using the following procedure:

  For each Attribute (Property or Relationship) included by the Entity Fragment:

  - If the target Entity does not include a matching Attribute (considering term expansion rules as mandated by clause 5.5.7), then such Attribute shall be appended to the target Entity.

  - If the target Entity already includes a matching Attribute (considering term expansion rules as mandated by clause 5.5.7):

    - If the Attribute (Property or Relationship) to be merged is represented in a simplified representation, the *type* of any pre-existing Attribute in the target entity shall be preserved.

    - If a common language tag is defined and a LanguageProperty Attribute to be merged is represented as a string, the pre-existing languageMap JSON object shall be preserved. The string value shall only replace the value associated to the language tag key found within the languageMap.

    - If a common "observedAt" timestamp is defined and an existing Attribute to be merged previously contained an "observedAt" sub-Attribute, the "observedAt" sub-Attribute is also updated using the common timestamp, unless the Entity Fragment itself contains an explicit updated value for the "observedAt" sub-Attribute.

    - If a *datasetId* is present in the Attribute included by the Entity Fragment:

      - If an Attribute instance in the target Entity has the same *datasetId*:

        - If overwrite is allowed and the Attribute value is not NGSI-LD Null, then the existing Attribute with the specified datasetId in the target Entity shall be merged with the new one supplied.

        - If overwrite is allowed and the Attribute value is NGSI-LD Null, then the existing Attribute with the specified datasetId in the target Entity shall be deleted.

        - If overwrite is not allowed, the existing Attribute with the specified datasetId in the target Entity shall be left untouched.

      - Otherwise the Attribute instance with the specified *datasetId* shall be appended to the target Entity.

    - If no *datasetId* is present in the Attribute included by the Entity Fragment, the default Attribute instance is targeted:

      - If the default Attribute instance is present:

        - If overwrite is allowed and the Attribute value is not NGSI-LD Null, then the existing Attribute in the target Entity shall be merged with the new one supplied.

- If overwrite is allowed and the Attribute value is NGSI-LD Null, then the existing Attribute with the specified datasetId in the target Entity shall be deleted.
- If overwrite is not allowed, the existing Attribute in the target Entity shall be left untouched.

- Otherwise if value is not NGSI-LD Null, the default Attribute instance shall be appended to the target Entity.

- If *type* is included in the Fragment and it includes Entity Type Names that are not yet in the target Entity, add them to the list of Entity Type Names of the target Entity.
- If *scope* is included in the Fragment and overwrite is allowed, the scope of the target Entity will become the one included in the Fragment. Otherwise, the Scopes in the Fragment that are not part of the value of *scope* of the target Entity will be appended to the value of the *scope* of the target Entity. If there is more than one Scope, the value of *scope* is represented as a JSON array containing all Scopes.

5.6.17.5 Output data

- A status code indicating whether all the Attributes were merged successfully.
- List of Attributes (Properties and/or Relationships) actually merged.

5.6.18 Replace Entity

5.6.18.1 Description

This operation allows the modification of an existing NGSI-LD Entity by replacing all of the Attributes (Properties or Relationships).

5.6.18.2 Use case diagram

A Context Producer can replace an entire Entity within an NGSI-LD system as shown in figure 5.6.18.2-1.

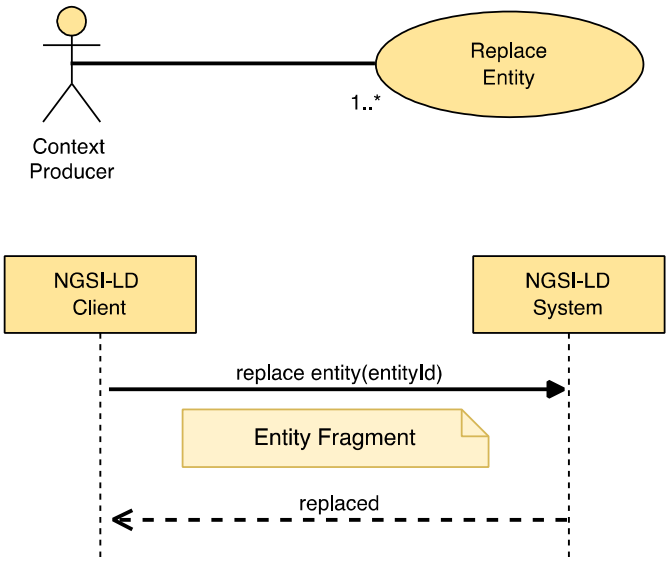

Figure 5.6.18.2-1: Replace Entity use case

5.6.18.3 Input data

- A URI representing the id of the Entity to be replaced (target Entity).
- A selector of Entity types as specified by clause 4.17 (optional).

- A JSON-LD document representing an NGSI-LD Entity.

5.6.18.4 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 NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally, and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- The behaviour defined in clause 5.5.4 on JSON-LD validation. NGSI-LD Nulls are not supported by this operation.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, Attributes from matching input data are forwarded. For each matching registration:
  - If the Replace Entity operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Replace Entity operation is not supported by the matched registration, this shall result in an error of type Conflict in case the complete Replace Entity operation failed, or in a partial success if some parts of it succeeded.
- The matching Attributes are then removed from the Fragment and not processed further.
- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing to matching endpoints.
- If the target Entity exists locally, completely replace the existing Entity with the same Entity id with the new Entity content provided. The system generated "createdAt" Temporal Properties of the Entity as defined in clause 4.8 remain unchanged.

5.6.18.5 Output data

- A status code indicating whether the Entity was replaced successfully.

5.6.19 Replace Attribute

5.6.19.1 Description

This operation allows the replacement of a single Attribute (Property or Relationship) within an NGSI-LD Entity**.**

5.6.19.2 Use case diagram

A Context Producer can carry out a replacement of an Attribute within an Entity within an NGSI-LD System as shown in figure 5.6.19.2-1.

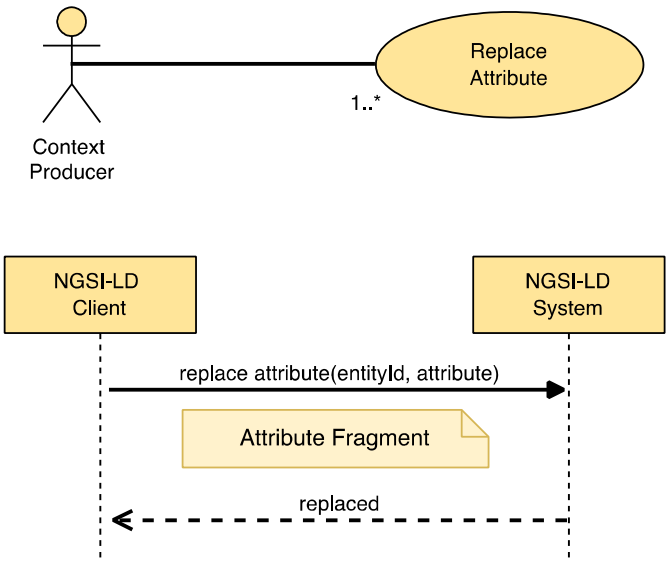

Figure 5.6.19.2-1: Attribute replace use case

5.6.19.3 Input data

- Entity Id (URI) of the concerned Entity, the target Entity.
- A selector of Entity types as specified by clause 4.17 (optional).
- Target Attribute (Property or Relationship) to be replaced, identified by a name.
- A JSON-LD document representing an NGSI-LD Attribute Fragment.

5.6.19.4 Behaviour

- If the target Entity id is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the target Attribute Name is not valid, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally, and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- The behaviour defined in clause 5.5.4 on JSON-LD validation.
- If an **exclusive** or **redirect** Context Source Registration matches against the input data, the input data is forwarded. For each matching registration:

  - If the Attribute Replace operation is supported by the matched registration, matching input data is forwarded to the Registration endpoint.
  - If the Attribute Replace operation is not supported by the matched registration, this shall result in an error of type *Conflict* in case the complete Attribute Replace operation failed, or in a partial success if some parts of it succeeded.

  No further processing is required.
- For any **inclusive** Context Source Registrations that exist and match against the remaining input data, that input data is also forwarded for remote processing to matching endpoints.
- 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 Attribute is *scope*, replace *scope* in the target Entity.

- If the target Entity does not contain the target Attribute:

  - as a default instance in case no *datasetId* is present;

  - as an instance with the specified *datasetId* if present;

  then this shall result in an error of type *ResourceNotFound* in case the complete Attribute Replace operation failed, or in a partial success if some parts of it succeeded.

- Completely replace the existing Attribute with the new Attribute content provided. The system generated "createdAt" Temporal Property as defined in clause 4.8 remains unchanged.

5.6.19.5 Output data

None.

5.6.20 Batch Entity Merge

5.6.20.1 Description

This operation allows modification of a batch of NGSI-LD Entities according to the JSON Merge Patch processing rules defined in IETF RFC 7396 [16] by adding new attributes (Properties or Relationships) or modifying or deleting existing attributes associated with an existing Entity.

5.6.20.2 Use case diagram

A Context Producer can merge a batch of Entities within an NGSI-LD system as shown in figure 5.6.20.2-1.

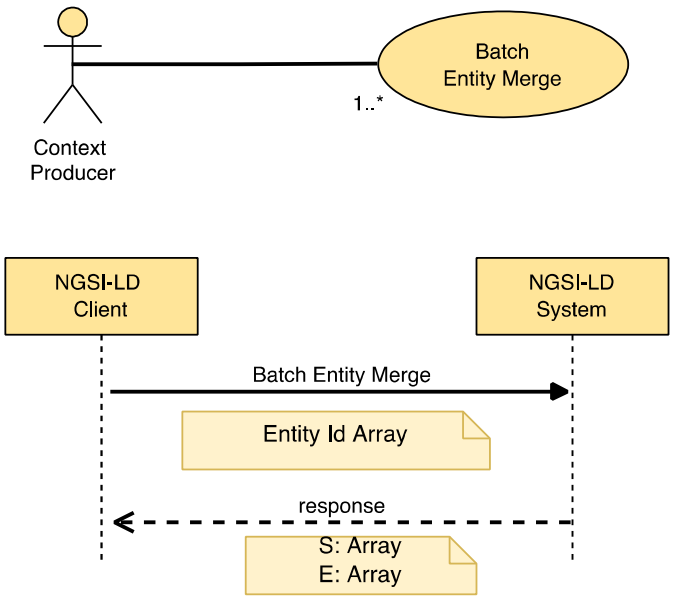

Figure 5.6.20.2-1: Merge a batch of Entities use case

5.6.20.3 Input data

- A JSON-LD Array containing one or more JSON-LD documents each one representing an Entity as mandated by clause 5.2.4. See clause 5.5.11.5 for information on behaviour when there is more than one instance of the same entity in the input Array.

5.6.20.4 Behaviour

Implementations shall exhibit the following behaviour:

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- Let S be an array which shall contain a list of Entity ids, one for each NGSI-LD Entity which was successfully processed. S shall be initialized as the empty array.
- Let E be an array which shall contain a list of *BatchEntityError* as defined by clause 5.2.17, one for each NGSI-LD Entity which resulted in error. E shall be initialized as the empty array.
- For each Context Source Registration CSR in the Context Registry:
  - Let IN be a copy of the original input array.
  - Remove from IN all Entities not matched by CSR and remove non-matching Attributes from the remaining Entities.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **exclusive** Context Source Registration, which is not CSR itself.
  - Remove all Attributes from the remaining Entities in IN for which there is a matching **redirect** Context Source Registration, unless CSR is a redirect Context Source Registration itself.
  - Remove all Entities without Attributes from IN.
  - If IN is empty, continue with the next Context Source Registration if there is any.
  - If the Batch Entity Merge operation is supported by CSR:
    - Forward the Batch Entity Merge request with IN as input Array.
    - Merge the returned list of Entities successfully created with S.
    - Merge the returned list of Entities in Error with E.
  - Otherwise, if the Merge Entity operation (clause 5.6.17) is supported by CSR:
    - For each Entity EN in the input array:
      - Forward a Merge Entity request for Entity EN.
      - Merge any successful result(s) for Entity EN merged with S.
      - Merge any error result(s) for Entity EN merged with E.
  - Otherwise:
    - In case CSR is an **exclusive** or **redirect** Context Source Registration, add an Error of type *Conflict* for each Entity in IN to E.
- For each of the NGSI-LD Entities included in the input Array execute the behaviour defined by clause 5.6.17, but limited to a local operation, as follows:
  - If the Entity was successfully merged (Attributes updated, appended or deleted), then add the corresponding Entity Id to the S array.
  - If the Entity merge failed, then a new *BatchEntityError* shall be added to E containing the failed Entity Id and the *ProblemDetails* associated.

5.6.20.5 Output data

- none (if all Entities already existed and are successfully merged); or
- the list of Entities successfully merged (S Array), and the list of Entities in error (E Array), if only some or none of the Entities have been successfully merged.

5.7 Context Information Consumption

5.7.1 Retrieve Entity

5.7.1.1 Description

This operation allows retrieving an NGSI-LD Entity.

5.7.1.2 Use case diagram

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

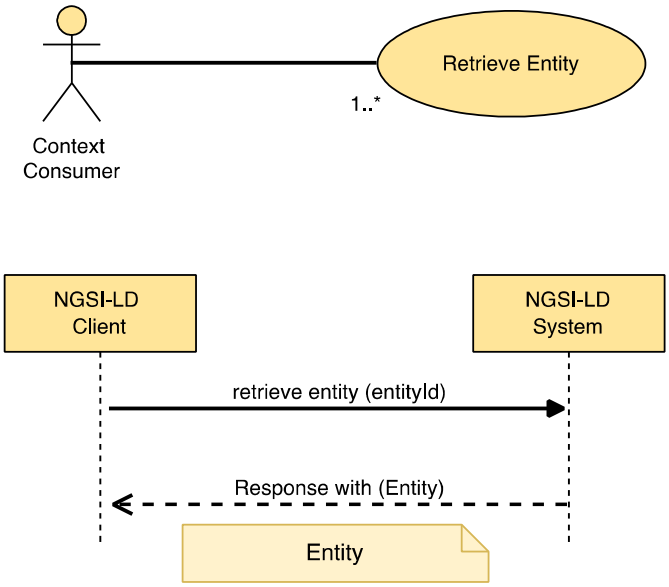

Figure 5.7.1.2-1: Retrieve Entity use case

5.7.1.3 Input data

- Entity Id (URI) of the Entity to be retrieved (target Entity).
- A selector of Entity types as specified by clause 4.17 (optional).
- List of Attribute (Properties or Relationships) Names to be retrieved (projection attributes) (optional).
- A language filter as defined by clause 4.15 (optional).
- An optional JSON-LD context.
- In the case of a GeoJSON representation:
  - The name of the **GeoProperty** attribute to use as the geometry for the GeoJSON representation as mandated by clause 4.5.16 (optional).
  - A datasetId specifying which instance of the value is to be selected if the **GeoProperty** value has multiple instances as defined by clause 4.5.5 (optional).

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 geometryProperty parameter is present and the Accept Header is not set to "application/geo+json", then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI), and where specified type, is equivalent held locally and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- The implementation shall retrieve any Attribute data held locally which is associated with the Entity defined by the Entity Id.
- For Context Source Registrations that match the Entity Id and support the retrieveEntity operation (see operations and operation groups in clause 4.20), implementations shall do the following:
  - For any **exclusive**, **redirect** and **inclusive** Context Source, the request is forwarded for remote retrieval by matching endpoints. and remote Attribute data for the Entity is received. It is then merged together according to the algorithm defined in clause 4.5.5.
  - For any **auxiliary** Context Source Registrations the remote Attribute data received is added to the payload only when an Attribute is not present in any of the Attribute data received elsewhere.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- If the optional Attribute list is present and the NGSI-LD endpoint does know about a matching Entity for the Entity Id, but this Entity does not have any of the Attributes in the Attribute list, then an error of type *ResourceNotFound* shall be raised.
- If the Accept Header is set to "application/json" or "application/ld+json, return a JSON-LD object representing the Entity as mandated by clause 5.2.4 and containing only the Attributes requested (if present).
- If the Accept Header is set to "application/geo+json", a GeoJSON Feature object representing the entity as mandated by clause 5.2.29 and containing only the Attributes requested (if present):
  - If the Prefer Header is omitted or set to "body=ld+json" then the Feature object will also contain an @context field.
  - If the Prefer Header is set to "body=json" the @context is set as a Link Header and removed from the Feature object.

5.7.1.5 Output data

A JSON-LD object representing the target Entity as mandated by clause 5.2.4 or a GeoJSON Feature as mandated by clause 5.2.29.

If any of the returned Attributes corresponds to a VocabularyProperty, the returned value shall be compacted according to the supplied @context.

If a language filter is specified and any of the returned Attributes corresponds to a LanguageProperty, the LanguageProperty in question shall be converted into a Property. The value of this Property shall correspond to the value of the string or strings from the matching key-value pair of the languageMap where the key matches the language filter. A non-reified subproperty `lang` shall be included in the response indicating the chosen language.

If no match can be made for a LanguageProperty then a single language shall be chosen, up to the implementation.

5.7.2 Query Entities

5.7.2.1 Description

This operation allows querying an NGSI-LD system.

5.7.2.2 Use case diagram

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

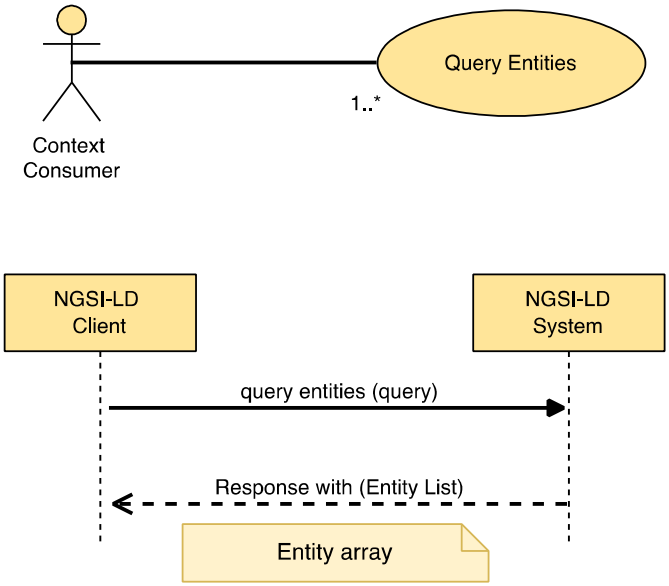

Figure 5.7.2.2-1: Query Entities use case

5.7.2.3 Input data

- A reference to a JSON-LD @context (optional).
- A selector of Entity types as specified by clause 4.17 (optional).
  Both type names (short hand string) and fully qualified type names (URI) are allowed in the selector.
- A list (one or more) of Entity identifiers (optional).
- A list (one or more) of Attribute names (called query projection attributes) (optional).
- An id pattern as a regular expression (optional).
- An NGSI-LD query (to filter out Entities by Attribute values) as per clause 4.9 (optional).
- An NGSI-LD geoquery (to filter out Entities by spatial relationships) as mandated by clause 4.10 (optional).
- In the case of GeoJSON representation:
  - The name of the **GeoProperty** attribute to use as the geometry for the GeoJSON representation as mandated by clause 4.5.16 (optional).
  - A datasetId specifying which instance of the value is to be selected if the **GeoProperty** value has multiple instances as defined by clause 4.5.5 (optional).
- A NGSI-LD Scope query (to filter out Entities based on their Scope) as mandated by clause 4.19 (optional).
- An NGSI-LD query (called context source filter, to filter out Context Sources by the values of properties that describe them) as per clause 4.9 (optional).
- A limit to the number of Entities to be retrieved. See clause 5.5.9.
- A specified language filter as per clause 4.15 (optional).
- A list (one or more) of Attribute names whose values shall be expanded to URIs prior to executing a query (optional).

It is not possible to retrieve a set of entities by only specifying desired Entity identifiers, without further specifying restrictions on the entities' types or attributes, either explicitly, via selector of Entity types or of Attribute names, or implicitly, within an NGSI-LD query or geoquery.

5.7.2.4 Behaviour

- At least one of the following input data shall be provided:

  a) *selector of Entity Types*;

  b) *list of Attribute names*;

  c) *NGSI-LD query*;

  d) *NGSI-LD geoquery*.

  If none of them is provided, then an error of type *BadRequestData* shall be raised (too wide query).

- If the list of Entity identifiers includes a URI which it is not valid, or the query, geoquery or context source filter are not syntactically valid (as per the referred clauses 4.9 and 4.10) an error of type *BadRequestData* shall be raised.

- If geometryProperty parameter is present and the Accept Header is not set to "application/geo+json", then an error of type BadRequestData shall be raised.

- Term to URI expansion of type and Attribute names shall be performed, as mandated by clause 5.5.7.

- If a list of Attribute names whose values shall be expanded to URIs has been supplied, the type coercion of those values to URIs shall be performed, as mandated by clause 5.5.7.

- Otherwise, implementations shall run a query that shall return an Entity Array containing all the Entities found locally, that meet **all** of the following conditions (given the respective parameter is provided):

  - id is equal to any of the id(s) passed as parameter;

  - the Entity Type Names match the selector of Entity Types (expanded) that is passed as parameter;

  - attribute matches any of the expanded attribute(s) in the list that is passed as parameter;

  - id matches the id pattern passed as parameter;

  - the filter conditions specified by the query are met (as mandated by clause 4.9);

  - the geospatial restrictions imposed by the geoquery are met (as mandated by clause 4.10); if there are multiple instances of the *GeoProperty* on which the geoquery is based, it is sufficient if any of these instances meets the geospatial restrictions;

  - if the Scope query is present, it shall match a present Entity Scope (as mandated by clause 4.19, for an example see annex C, clause C.5.15);

  - if the Attribute list is present, in order for an Entity to match, it shall contain at least one of the Attributes in the projection Attribute list.

- For Context Source Registrations that match the query and support the "queryEntity" operation (see operations and operation groups in clause 4.20), implementations shall do the following:

  - For any **exclusive**, **redirect** and **inclusive** Context Source Registrations, the request is forwarded for remote querying by matching endpoints. The result of each remote query is an Entity Array. The Entity Arrays are then merged together with the locally queried result according to the algorithm defined in clause 4.5.5.

  - For any **auxiliary** Context Source Registrations, the request is forwarded for remote querying by matching endpoints. Data from the Entity Array received is added to the payload only when an Attribute is not already present in the merged Entity Arrays are received elsewhere.

- Pagination logic shall be in place as mandated by clause 5.5.9.

- If in the process of obtaining the query result it is necessary to issue a Context Source discovery operation, the same Context Source filter input parameter (if present) shall be propagated.
- If the Accept Header is set to "application/json" or "application/ld+json, a JSON-LD array is returned, representing the Entities as mandated by clause 5.2.4 and containing only the Attributes requested (if present).
- If the Accept Header is set to "application/geo+json", the response shall be a GeoJSON FeatureCollection as mandated by clause 5.2.30, with each Feature within the FeatureCollection containing only the Attributes requested (if present):
    - If the Prefer Header is omitted or set to "body=ld+json" then the FeatureCollection will also contain an @context field.
    - If the Prefer Header is set to "body=json" the @context is sent as a Link Header and removed from the FeatureCollection object.

5.7.2.5 Output data

A JSON-LD array representing the matching entities as defined by clause 5.2.4 or in the case of GeoJSON requests a FeatureCollection as mandated by clause 5.2.30. 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.

If any of the returned Attributes corresponds to a VocabularyProperty, the returned value shall be compacted according to the supplied @context.

If a language filter is specified and any of the returned Attributes corresponds to a LanguageProperty, the LanguageProperty in question shall be converted into a Property. The value of this Property shall correspond to the value of the string or strings from matching key-value pair of the languageMap where the key matches the language filter. A non-reified subproperty `lang` shall be included in the response indicating the chosen language.

If no match can be made for a LanguageProperty, then the default identified by the JSON-LD *"@none* shall be chosen if present, otherwise the choice of a single language is up to the implementation.

5.7.3 Retrieve Temporal Evolution of an Entity

5.7.3.1 Description

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

5.7.3.2 Use case diagram

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

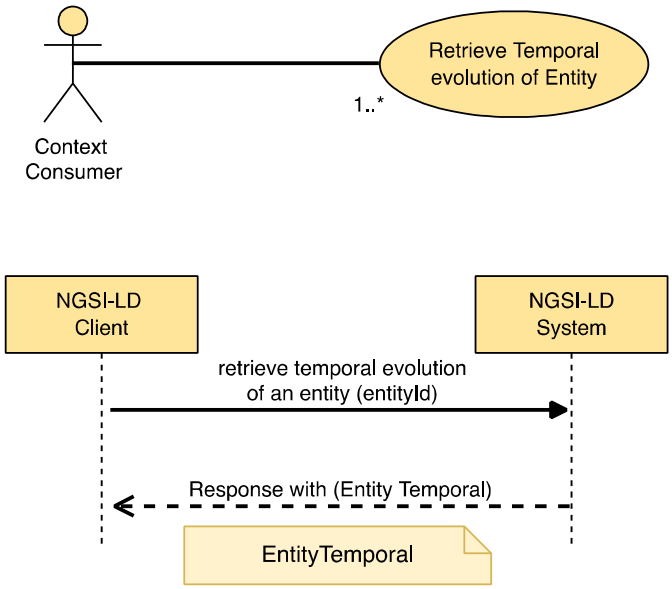

Figure 5.7.3.2-1: Retrieve temporal evolution of Entity use case

5.7.3.3 Input data

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

5.7.3.4 Behaviour

- If the Entity Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint does not know about the target Entity, because there is no existing Entity whose id (URI) is equivalent held locally, and no matching registrations apply, then an error of type *ResourceNotFound* shall be raised.
- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- The *lastN* parameter refers to a number, *n*, of Attribute instances which shall correspond to the last *n* timestamps (in descending ordering) of the temporal property (by default *observedAt*) within the concerned temporal interval.
- Let S be the Temporal Representation of the Entity as mandated by clause 5.2.20 with the specified Entity Id as it is available locally. S is empty in case no Temporal Representation of the Entity is available locally.
- From S, select only those Attribute instances (corresponding to the Attributes specified by the query or all if none are specified) match the temporal restrictions imposed by the temporal query (as mandated by clause 4.11), i.e. if the time series, for all the concerned Attributes of an Entity, does not include data corresponding to the temporal query interval, then such Entity shall be removed from S, thus it shall not appear in the final result set. Let S1 be this new subset.

- For Context Source Registrations that match the Entity Id and support the retrieveTemporal operation (see operations and operation groups in clause 4.20), implementations shall do the following:

  - For any **exclusive**, **redirect** and **inclusive** Context Source, the request is forwarded for remote retrieval by matching endpoints. and remote Attribute data for the Entity is received. The result is then merged together with S1 according to the algorithm defined in clause 4.5.5.

  - For any **auxiliary** Context Source Registrations the remote Attribute data received is added to S1 only when the Attribute instance, whose value of the *timeproperty*, which is used for the temporal query (*observedAt* as default), is not present in any of the Attribute instances received from elsewhere.

- From the set of Attribute Instances that are in S1, include in their temporal representation only the Attribute instances (up to *lastN*) corresponding to the query's projection Attributes, or aggregated values of Attribute instances (if aggregated temporal representation is requested).

  If an aggregated temporal representation is requested and any of the requested Attributes is not eligible for at least one of the aggregation methods specified in the request parameters, then an error of type *InvalidRequest* shall be raised.

5.7.3.5 Output data

A JSON-LD object representing the Temporal Representation of the target Entity as mandated by clause 5.2.20.

5.7.4 Query Temporal Evolution of Entities

5.7.4.1 Description

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

5.7.4.2 Use case diagram

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

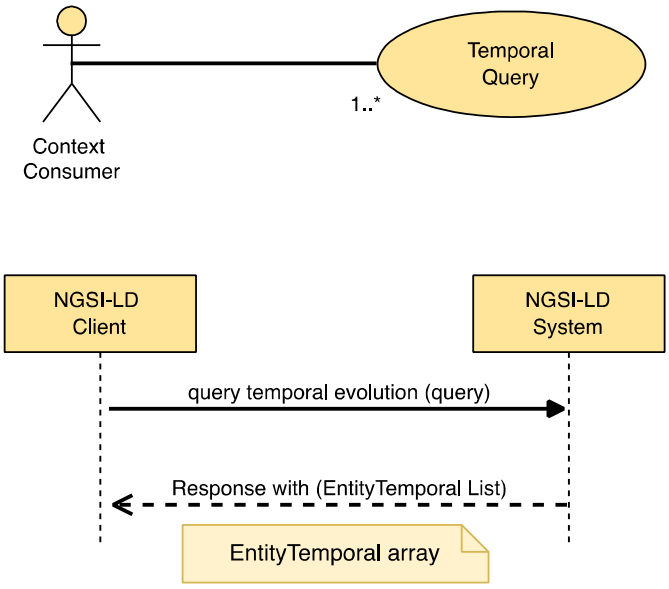

Figure 5.7.4.2-1: Temporal query use case

5.7.4.3 Input data

- An NGSI-LD temporal query as mandated by clause 4.11.
- A reference to a JSON-LD @context (optional).
- A selector of Entity types as specified by clause 4.17 (optional).
  Both type name (short hand string) and fully qualified type name (URI) are allowed.
- A list (one or more) of Entity identifiers (optional).
- A list (one or more) of Attribute names (called query projection attributes) (optional).
- An id pattern as a regular expression (optional).
- An NGSI-LD query (to filter out Entities by Attribute values) as per clause 4.9 (optional).
- An NGSI-LD geoquery (to filter out Entities by spatial relationships) as mandated by clause 4.10 (optional).
- A NGSI-LD Scope query (to filter out Entities based on their Scope) as mandated by clause 4.19 (optional).
- An NGSI-LD query (called context source filter, to filter out Context Sources by the values of properties that describe them) as per clause 4.9 (optional).
- A limit to the number of Entities to be retrieved. See clause 5.5.9.
- A parameter (lastN) conveying that only the last N instances (per Attribute) within the concerned temporal interval shall be retrieved (optional).
- A specified language filter as per clause 4.15 (optional).
- A list (one or more) of Attribute names whose values shall be expanded to URIs prior to executing a query (optional).

It is not possible to retrieve a set of entities by only specifying desired Entity identifiers, without further specifying restrictions on the entities' types or attributes, either explicitly, via selector of Entity types or of Attribute names, or implicitly, within an NGSI-LD query or geoquery.

5.7.4.4 Behaviour

- If a temporal query is not provided then an error of type *BadRequestData* shall be raised.
- At least one of the following input data shall be provided:

  a) *selector of Entity Types*;

  b) *list of Attribute names*;

  c) *NGSI-LD query*;

  d) *NGSI-LD geoquery*.

  If none of them is provided, then an error of type *BadRequestData* shall be raised (too wide query).
- If the list of Entity identifiers includes a URI which it is not valid, or the query, geoquery or context source filter 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.
- If a list of Attribute names whose values shall be expanded to URIs has been supplied, the type coercion of those values to URIs shall be performed, as mandated by clause 5.5.7.

- The *lastN* parameter refers to a number, *n*, of Attribute instances which shall correspond to the last *n* timestamps (in descending ordering) of the temporal property (by default *observedAt*) within the concerned temporal interval.

- Otherwise, implementations shall run a query that shall return the temporal evolution of the matching Entities; the logical steps to select the final result set of Entities, and the Attribute instances included as part of their temporal representation, are enumerated as follows:

  - Let S be the set of selected Entities i.e. the query result set.

  - If id(s) is provided, keep in S only those Entities whose id is equivalent to any of the id(s) passed as parameter.

  - If an id pattern is provided, keep in S only those Entities whose id matches the id pattern.

  - If a selector of Entity Types is provided, keep in S only those Entities whose Entity Type Names match the selector of Entity Types.

  - From S, select only those Entities any of whose Attribute instances (corresponding to the Attributes specified by the query or all if none are specified) match the temporal restrictions imposed by the temporal query (as mandated by clause 4.11); i.e. if the time series, for all the concerned Attributes of an Entity, does not include data corresponding to the temporal query interval, then such Entity shall be removed from S, thus it shall not appear in the final result set. Let S1 be this new subset.

  - If a values filter query is provided, from S1, select those Entities whose Attribute instances (during the interval defined by the temporal query) meet the matching conditions specified by the query (as mandated by clause 4.9), i.e. the values filter query shall be checked against all the Attribute instances resulting from the initial filtering performed by the temporal query. Let S2 be this new subset.

  - If no values filter query is provided, then S2 is equal to S1.

  - If geoquery is present, from S2, select those Entities whose *GeoProperty* instances meet the geospatial restrictions imposed by the geoquery (as mandated by clause 4.10); those geospatial restrictions shall be checked against the *GeoProperty* instances that are within the interval defined by the temporal query. Let S3 be this new subset.

  - If no geoquery is provided, then S3 is equal to S2.

  - If the Scope query is present, from S3, select those Entities whose Entity Scope instances match the Scope query (as mandated by clause 4.19, for an example see annex C, clause C.5.16). Let S4 be the new subset.

  - If no Scope query is provided, then S4 is equal to S3.

- For Context Source Registrations that match the query and support the "queryTemporal" operation (see operations and operation groups in clause 4.20), implementations shall do the following:

  - For any **exclusive**, **redirect** and **inclusive** Context Source Registrations that match against the query, the request is forwarded for remote querying by matching endpoints. The result of each remote query is an Entity Array. The returned result is then merged into S4 according to the algorithm defined in clause 4.5.5.

  - For any **auxiliary** Context Source Registrations that match against the query, the request is forwarded for remote querying by matching endpoints. Data from the Entity Array received is merged only into S4 when an Attribute instance, whose value of the timeproperty used for the temporal query, is not already present in S4.

- From the set of Entities that are in S4, include in their temporal representation only the Attribute instances (up to *lastN*) corresponding to the query's projection Attributes, or aggregated values of Attribute instances (if aggregated temporal representation is requested), and which meet the temporal, query and geoquery restrictions.

  If an aggregated temporal representation is requested and any of the requested Attributes is not eligible for at least one of the aggregation methods specified in the request parameters, then an error of type *InvalidRequest* shall be raised.

- Pagination logic shall be in place as mandated by clause 5.5.9.
- If in the process of obtaining the query result it is necessary to issue a Context Source discovery operation, the same Context Source filter input parameter (if present) shall be propagated.

5.7.4.5 Output Data

A JSON-LD array representing the matching entities as defined by clause 5.2.21 and selected according to the behaviour described by clause 5.7.4.4.

5.7.5 Retrieve Available Entity Types

5.7.5.1 Description

This operation allows retrieving a list of NGSI-LD entity types for which entity instances exist within the NGSI-LD system.

5.7.5.2 Use case diagram

A context consumer can retrieve a list of NGSI-LD entity types from the system as shown in figure 5.7.5.2-1.

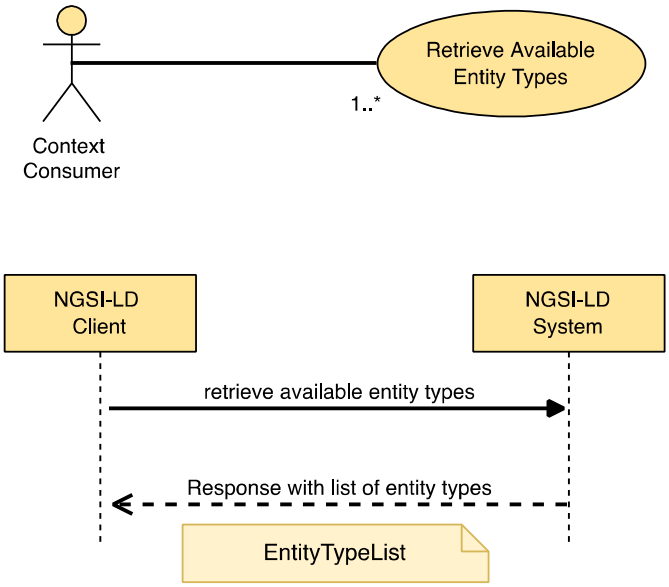

Figure 5.7.5.2-1: Retrieve Available Entity Types use case

5.7.5.3 Input data

- An optional JSON-LD context.

5.7.5.4 Behaviour

- Return a JSON-LD object representing the list of entity types, as mandated by clause 5.2.24, for which entity instances exist within the NGSI-LD system. See clause 5.7.11 for architecture-related implementation aspects.

5.7.5.5 Output data

A JSON-LD object representing the list of available entity types, as mandated by clause 5.2.24.

5.7.6 Retrieve Details of Available Entity Types

5.7.6.1 Description

This operation allows retrieving a list with a detailed representation of NGSI-LD entity types for which entity instances exist within the NGSI-LD system. The detailed representation includes the type name (as short name if available in the provided @context) and the attribute names that existing instances of this entity type have.

5.7.6.2 Use case diagram

A context consumer can retrieve a list with a detailed representation of NGSI-LD entity types from the system as shown in figure 5.7.6.2-1.

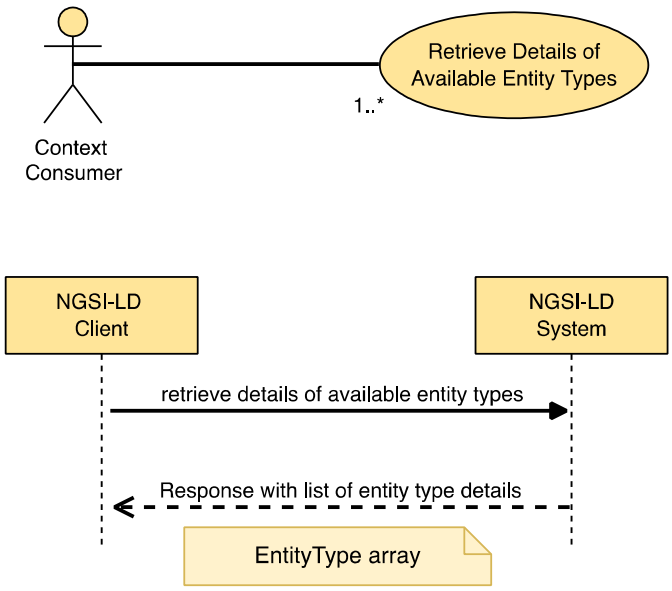

Figure 5.7.6.2-1: Retrieve Details of Available Entity Types use case

5.7.6.3 Input data

- An optional JSON-LD context.

5.7.6.4 Behaviour

- Return a list of JSON-LD objects representing the details of available entity types as mandated by clause 5.2.25 for which entity instances exist within the NGSI-LD system. See clause 5.7.11 for architecture-related implementation aspects.

5.7.6.5 Output data

A list of JSON-LD objects representing the details of available entity types as mandated by clause 5.2.25.

5.7.7 Retrieve Available Entity Type Information

5.7.7.1 Description

This operation allows retrieving detailed entity type information about a specified NGSI-LD entity type for which entity instances exist within the NGSI-LD system. The detailed representation includes the type name (as short name if available in the provided @context), the count of available entity instances and details about attributes that existing instances of this entity type have, including their name (as short name if available in the provided @context) and a list of types the attribute can have (e.g. Property, Relationship or GeoProperty).

5.7.7.2 Use case diagram

A context consumer can retrieve a detailed representation of a specified NGSI-LD entity type from the system as shown in figure 5.7.7.2-1.

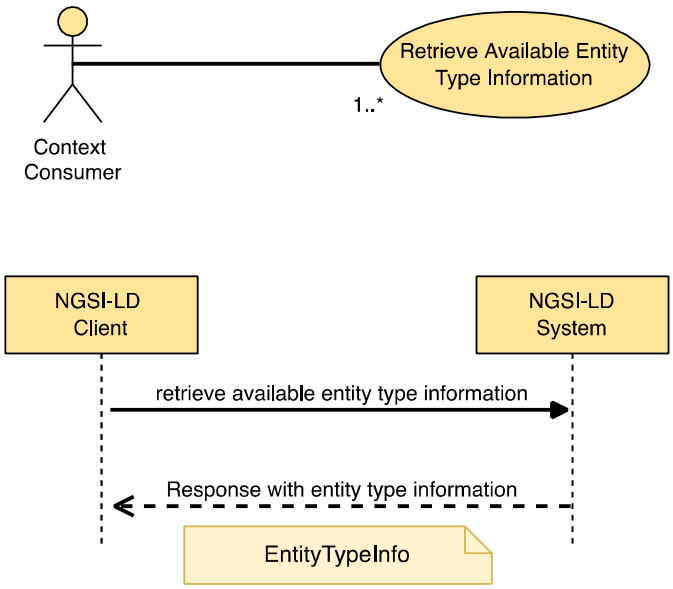

Figure 5.7.7.2-1: Retrieve Available Entity Type Information use case

5.7.7.3 Input data

- Entity type name for which detailed information is to be retrieved.
- An optional JSON-LD context.

5.7.7.4 Behaviour

- Return a JSON-LD object representing the details of the specified entity type as mandated by clause 5.2.26, for which instances exist within the NGSI-LD system. See clause 5.7.11 for architecture-related implementation aspects.

5.7.7.5 Output data

A JSON-LD object representing the details of the specified entity type as mandated by clause 5.2.26.

5.7.8 Retrieve Available Attributes

5.7.8.1 Description

This operation allows retrieving a list of NGSI-LD attributes that belong to entity instances existing within the NGSI-LD system.

5.7.8.2 Use case diagram

A context consumer can retrieve a list of NGSI-LD attributes from the system as shown in figure 5.7.8.2-1.

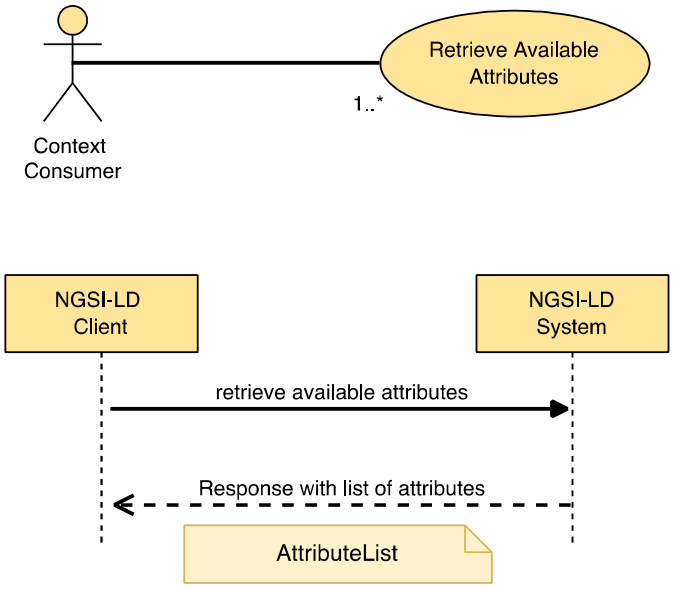

Figure 5.7.8.2-1: Retrieve Available Attributes use case

5.7.8.3 Input data

- An optional JSON-LD context.

5.7.8.4 Behaviour

- Return a JSON-LD object representing the list of attributes as mandated by clause 5.2.27 that belong to entity instances existing within the NGSI-LD system. See clause 5.7.11 for architecture-related implementation aspects.

5.7.8.5 Output data

A JSON-LD object representing the list of available attributes as mandated by clause 5.2.27.

5.7.9 Retrieve Details of Available Attributes

5.7.9.1 Description

This operation allows retrieving a list with a detailed representation of NGSI-LD attributes that belong to entity instances existing within the NGSI-LD system. The detailed representation includes the attribute name (as short name if available in the provided @context) and the type names for which entity instances exist that have the respective attribute.

5.7.9.2 Use case diagram

A context consumer can retrieve a list with a detailed representation of NGSI-LD attributes from the system as shown in figure 5.7.9.2-1.

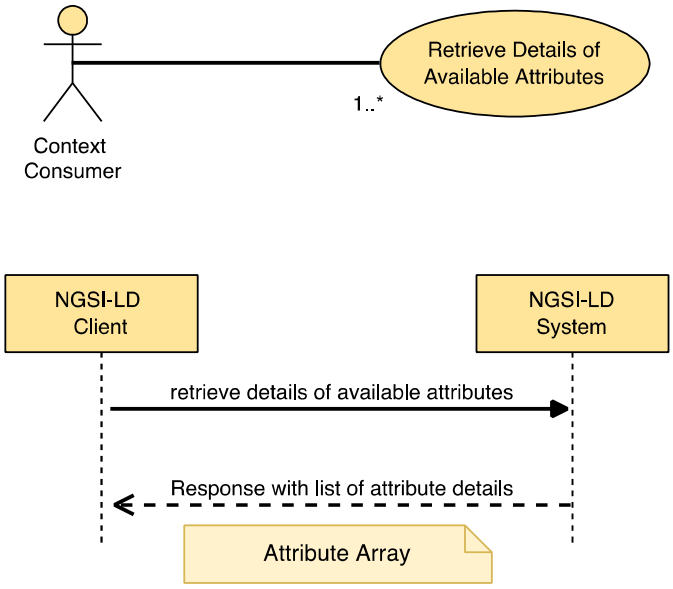

Figure 5.7.9.2-1: Retrieve Details of Available Attributes use case

5.7.9.3 Input data

- An optional JSON-LD context.

5.7.9.4 Behaviour

- Return a list of JSON-LD objects representing the details of available attributes as mandated by clause 5.2.28 (restricted to the elements id, type, attributeName and typeNames) that belong to entity instances existing within the NGSI-LD system. See clause 5.7.11 for architecture-related implementation aspects.

5.7.9.5 Output data

A list of JSON-LD objects representing the details of available attributes as mandated by clause 5.2.28 (restricted to the elements id, type, attributeName and typeNames).

5.7.10 Retrieve Available Attribute Information

5.7.10.1 Description

This operation allows retrieving detailed attribute information about a specified NGSI-LD attribute that belongs to entity instances existing within the NGSI-LD system. The detailed representation includes the attribute name (as short name if available in the provided @context) and the type names for which entity instances exist that have the respective attribute, a count of available attribute instances and a list of types the attribute can have (e.g. Property, Relationship or GeoProperty).

5.7.10.2 Use case diagram

A context consumer can retrieve a list with a detailed representation of NGSI-LD attributes from the system as shown in figure 5.7.10.2-1.

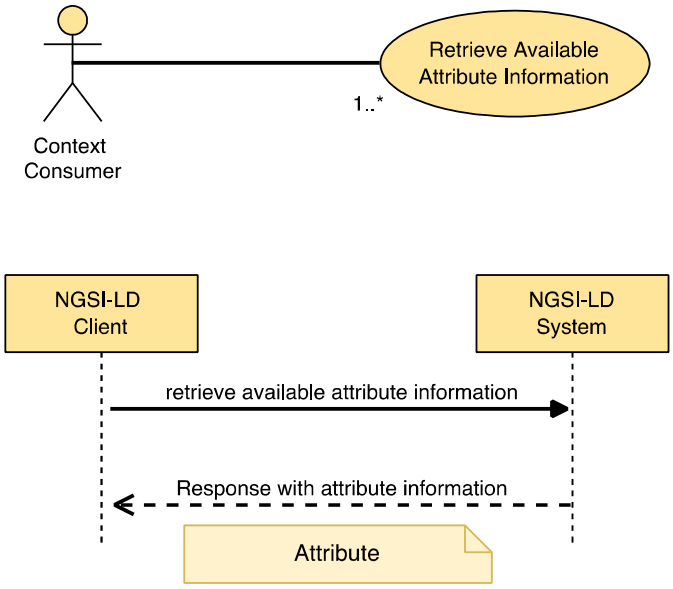

Figure 5.7.10.2-1: Retrieve Available Attribute Information use case

5.7.10.3 Input data

- Name of the attribute for which detailed information is to be retrieved.
- An optional JSON-LD context.

5.7.10.4 Behaviour

- Return a JSON-LD object representing the details of available attributes as mandated by clause 5.2.28 that belong to entity instances existing within the NGSI-LD system. See clause 5.7.11 for architecture-related implementation aspects.

5.7.10.5 Output data

A JSON-LD object representing the details of available attributes as mandated by clause 5.2.28.

5.7.11 Architecture-related aspects of retrieval of entity types and attributes

Retrieving information about available types or attributes can be an expensive operation depending on the scale and architectural design decisions of the NGSI-LD system. This is in particular the case for retrieving the information about all available entity types and attributes related to all entity information available in an NGSI-LD system. Especially in the case of distributed architecture (clause 4.3.3) and federated architecture (clause 4.3.4) checking all entities can be so expensive that it can become practically infeasible.

Therefore, implementations may only take into account only information that is available or can be derived from a local datastore and the Context Registry, when implementing the retrieval of available entity types and attributes, as described in clauses 5.7.5, 5.7.6, 5.7.7, 5.7.8, 5.7.9 and 5.7.10. Context registrations do not always reflect which entity instances are actually available from a Context Source at a particular point in time, but only which entity instances are possibly available from a Context Source, thus in this case the information about available entity types and attributes is to be interpreted as "possibly available". Also, context registrations can have different granularities, i.e. they possibly only contain entity type or attribute information, and thus the provided information about available entity types and attributes is possibly incomplete as a result. In particular the attributeNames in the EntityType data structure (clause 5.2.25), the attributeDetails in the EntityTypeInfo data structure (clause 5.2.26), and the attributeTypes and typeNames in the Attribute data structure (clause 5.2.27) may be provided as empty arrays if the information is not included in the respective context registration. Implementations may also provide estimates for the entity count or attribute count instead of the accurate count.

As an alternative to relying on local information only, the request can be forwarded to all Context Sources which support the respective operation according to the Context Source Registration describing them. In this case the returned lists are merged with the local list of entity types before returning them. This approach is more expensive but leads to a more accurate result.

5.8 Context Information Subscription

5.8.1 Create Subscription

5.8.1.1 Description

This operation allows creating a new subscription.

5.8.1.2 Use case diagram

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

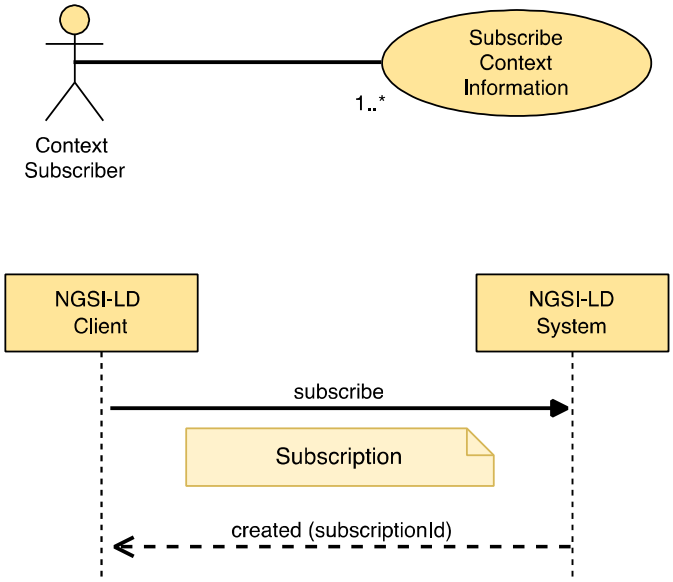

Figure 5.8.1.2-1: Create subscription use case

5.8.1.3 Input data

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

5.8.1.4 Behaviour

- If the data types, cardinalities and restrictions expressed by clause 5.2.12 are not met, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD endpoint already knows about this Subscription, as there is an existing Subscription whose id (URI) is equivalent, an error of type *AlreadyExists* shall be raised.
- If the subscription document does not include a Subscription identifier, a new identifier (URI) shall be automatically generated by the implementation.

- Then, implementations shall add a new Subscription. The parameters of the created Subscription shall be configured as follows:
  - The Subscription expiration date shall be equal to the value of the *expiresAt* 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 *expiresAt* member the Subscription shall be considered as perpetual.
  - If the value of the *isActive* field is not included or is *true* then the initial status of the Subscription shall be set to "active".
  - If the value of the *isActive* field is *false*, then the initial status of the Subscription shall be set to "paused".
  - If present, the subscribed entities shall be those matching the conditions expressed under the *EntitySelector*, as defined in clause 5.2.33.
  - Watched Attributes shall be those Attributes (subject to clause 5.5.7 Term to URI expansion) pertaining to the subscribed entities (if present) and conveyed through the *watchedAttributes* member. Watched Attributes are those that trigger a new notification when they are changed. A non-present *watchedAttributes* member means that all Attributes shall be watched. If no subscribed entities have been specified, all entities with attributes matching at least one member of *watchedAttributes* are subscribed to.
  - The @context to be used for sending Notifications related to this Subscription shall be the one specified in the *jsonldContext* field. If not present, the *jsonldContext* field shall be initialized with the @context applicable for the Subscription (if @context is also not present in the Subscription, see clause 5.5.5). When the remote JSON-LD @context referenced by the *jsonldContext* field is not available implementations shall raise an error of type *LdContextNotAvailable*. If the remote JSON-LD @context referenced by the *jsonldContext* field is invalid, implementations shall raise an error of type *BadRequestData*.
  - Based on the content of the Subscription, a Context Source Registration Subscription shall be created (clause 5.11.2). The mapping of the *id* of the Subscription to the returned *subscriptionId* of the Context Source Registration Subscription shall be stored to enable updating or deleting the Context Source Registration Subscription in case of changes to the Subscription.
- 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.
- Each time a Context Source Notification with the *subscriptionId* of the previously created Context Source Registration Subscription is received, implementations shall do the following:
  - For any **exclusive**, **redirect** and **inclusive** Context Source Registration received as part of the notification, implementation shall do the following depending on the *triggerReason*:
    - If the *triggerReason* is *"newlyMatching"* and the Context Source Registration indicates support for the Create Subscription operation, a copy of the original Subscription shall be reduced to what is matched by the registration *information* and forwarded to the Context Source as a new Subscription where the notification endpoint is set to that of the local Broker. The mapping of the received *subscriptionId* with the own Subscription identifier is stored to enable forwarding received notifications to the original subscriber. In addition, a mapping of the id of the Context Source Registration to the received *subscriptionId* is stored, to enable updating or deleting the subscription in case of changes.
    - If the *triggerReason* is *"updated"* and the Context Source Registration indicates support for the Update Subscription operation, an update of the original Subscription, reduced to what is matched by the registration *information*, shall be forwarded to the Context Source with the *subscriptionId* related to the Context Source Registration. As an optimization, an implementation may keep the originally forwarded Context Source Registration and compare with the new one to only forward the update, if there was a relevant change.

- If the *triggerReason* is *"noLongerMatching"* and the Context Source Registration indicates support for the delete Subscription operation, a delete Subscription shall be forwarded to the Context Source with the *subscriptionId* related to the Context Source Registration.

- Implementations shall ensure that, when the Subscription expiration date is due, the status of the Subscription changes automatically to *expired*, so that notifications will no longer be sent.

5.8.1.5 Output data

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

5.8.2 Update Subscription

5.8.2.1 Description

This operation allows updating an existing subscription.

5.8.2.2 Use case diagram

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

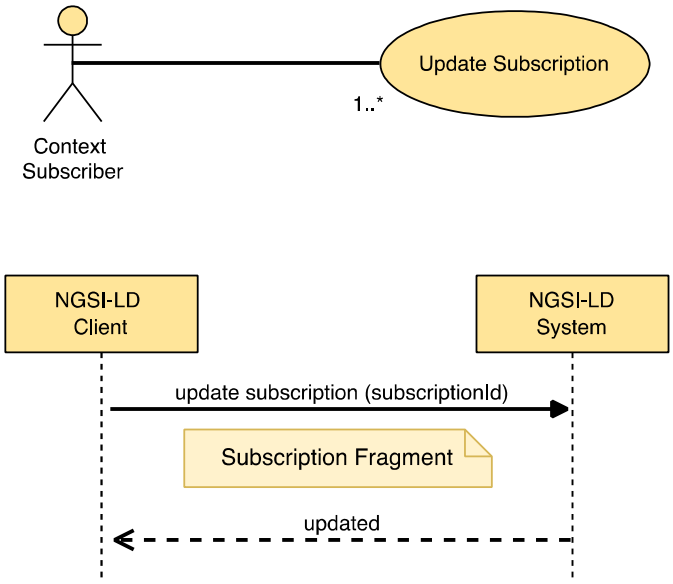

Figure 5.8.2.2-1: Update subscription use case

5.8.2.3 Input data

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

5.8.2.4 Behaviour

- If the Subscription id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the NGSI-LD System does not know about the target Subscription, because there is no existing Subscription whose id (URI) is equivalent, an error of type *ResourceNotFound* shall be raised.
- Execute the behaviour defined in 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.
- If the *jsonldContext* field is present and the referenced JSON-LD @context is not available, implementations shall raise an error of type *LdContextNotAvailable*. If the referenced JSON-LD @context is invalid, implementations shall raise an error of type *BadRequestData*.
- Then, implementations shall modify the target Subscription as mandated by clause 5.5.8.
- Finally, the following extra behaviour shall be observed when updating Subscriptions:
  - If *isActive* is equal to *true* and *expiresAt* is not present, then *status* shall be updated to "active", if and only if, the previous value of *status* was different than "expired".
  - If *isActive* is equal to *true* and *expiresAt* corresponds to a *DateTime* in the future, then *status* shall be updated to "active".
  - If *isActive* is equal to *false* and *expiresAt* is not present, then status shall be updated to "paused", if and only if, the previous value of *status* was different than "expired".
  - If only *expiresAt* is included and refers to a *DateTime* in the future, then *status* shall be updated to "active", if and only if the previous value of *status* was "expired".
  - If *expiresAt* is included but referring to a *DateTime* in the past, then a *BadRequestData* error shall be raised, regardless the value of *isActive*.
  - Based on the mapping of the Subscription to its respective Context Source Registration Subscription (see clause 5.8.1.4), that Context Source Registration Subscription shall be updated (clause 5.11.3).

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.

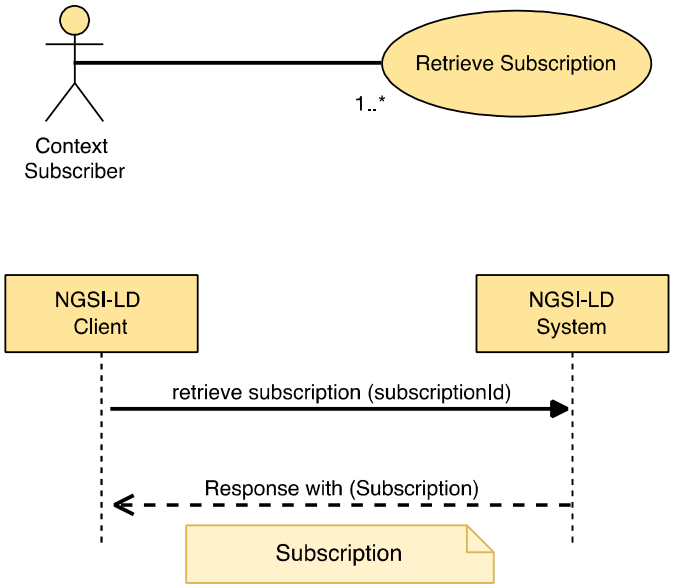

Figure 5.8.3.2-1: Retrieve subscription use case

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 subscriptions and obtain the subscription data to be returned to the caller.

5.8.3.5 Output data

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

5.8.4 Query Subscriptions

5.8.4.1 Description

This operation allows querying existing Subscriptions.

5.8.4.2 Use case diagram

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

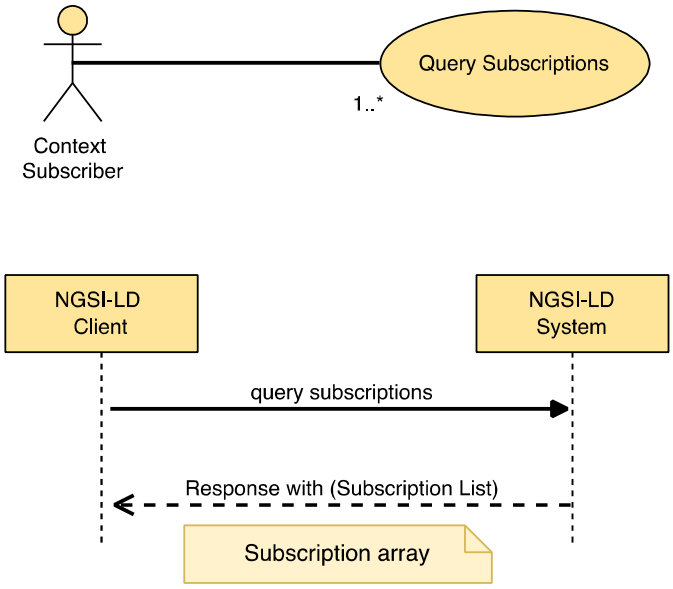

Figure 5.8.4.2-1: Query subscriptions use case

5.8.4.3 Input data

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

5.8.4.4 Behaviour

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

5.8.4.5 Output data

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

5.8.5 Delete Subscription

5.8.5.1 Description

This operation allows deleting an existing subscription.

5.8.5.2 Use case diagram

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

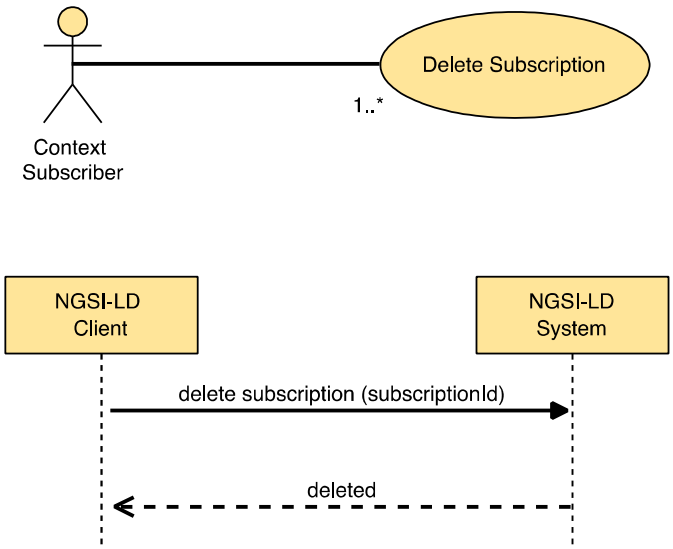

Figure 5.8.5.2-1: Delete subscription use case

5.8.5.3 Input data

- A subscription identifier (URI).

5.8.5.4 Behaviour

- If the subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the subscription id provided does not correspond to any existing subscription in the system then an error of type *ResourceNotFound* shall be raised.
- Otherwise implementations shall delete the Subscription and no longer perform notifications concerning such Subscription.
- Using the mapping of the own Subscription identifier to each of the *subscriptionId* of a subscription to a Context Source, a delete Subscription shall be forwarded to each such Context Source, if the delete Subscription operation is supported as indicated in the corresponding Context Source Registration:
  - Based on the mapping of the Subscription to its respective Context Source Registration Subscription (see clause 5.8.1.4), that Context Source Registration Subscription shall be deleted (clause 5.11.6).

5.8.5.5 Output data

None.

5.8.6 Notification behaviour

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

- Notifications shall only be sent if and only if the status of the corresponding subscription ("subscription.status") is *active*, i.e. not *paused* nor *expired*.
- If a Subscription defines a *timeInterval* member, a Notification shall be sent periodically, when the time interval (in seconds) specified in such value field is reached, regardless of Attribute changes. The notification message shall include all the subscribed Entities that match the query, geoquery and Scope query conditions. If none of query, geoquery and Scope query are defined, then all subscribed Entities shall be included.

- If a Subscription does not define a *timeInterval* term, the notification shall be sent whenever there is a change in the watched Attributes. An Attribute is considered to change when any of the members (including children) in its corresponding JSON-LD node is updated with a value different than the existing one. The notification message shall include all the subscribed Entities that changed and that match (as mandated by clauses 4.9 and 4.10) the query and geoquery conditions. If query or geoquery are not defined then all subscribed Entities that changed shall be included. If, for an Entity, there are multiple instances of the *GeoProperty* on which the geoquery is based, it is sufficient if any of these instances meets the geospatial restrictions. Finally, if a Context Source filter is defined, then only the subscribed Entities whose origin Context Source matches the referred filter shall be included.

- If a Notification with a *subscriptionId* is received that has a mapping to a local Subscription identifier, the Notification shall be forwarded to the Notification endpoint specified in the local Subscription, using this local Subscription identifier instead of the *subscriptionId* received.

- A Notification shall be sent as follows:

  - The structure of the notification message shall be as mandated by clause 5.3.1.

  - The @context to be used is the one specified in the jsonldContext field of the corresponding Subscription.

  - The Attributes included (Properties or Relationships) shall be those specified by the *notification.attributes* member in the Subscription data type (clause 5.2.12). Term to URI expansion shall be observed (clause 5.5.7). The absence of the *notification.attributes* member of a Subscription means that all Attributes shall be included. If the notification was triggered by the deletion of an Entity and the *notification.showChanges* member is not set to *true*, only the *deletedAt* system property shall be provided. In case *notification.sysAttrs* is set to *true*, also *createdAt* and *modifiedAt* shall be provided.

  - If an Attribute has been deleted, only the name of the attribute as key and the URI *"urn:ngsi-ld:null"* as value shall be provided, unless more information is required. The latter is the case, if:

    - a *datasetId* needs to be provided;

    - the *notification.sysAttrs* is set to true and thus the system generated sub-attributes have to be provided;

    - *notification.showChanges* is set to true and thus a previous value or object has to be provided.

    In all such cases, a JSON object with all the required information is provided, where the *value* or the *object* is set to the URI *"urn:ngsi-ld:null"* respectively or, in case of a Language Property, the languageMap is set to *{"@none": "urn:ngsi-ld:null"}*.

  - 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 and including any optional *endpoint.receiverInfo* defined by clause 5.2.22) 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 or GeoJSON by means of the *endpoint.accept* member.

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

  - The *notification.lastNotification* member shall be updated with a timestamp representing the current date and time.

  - If the response to the notification request is 200 OK then implementations shall:

    - Update *notification.lastSuccess* with a timestamp representing the current date and time.

    - Update *notification.status* to "ok".

  - If the response to the notification request is different than 200 OK then implementations shall:

    - Update *notification.lastFailure* with a timestamp representing the current date and time.

    - Update *notification. status* to "failed".

5.9 Context Source Registration

5.9.1 Introduction

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

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

5.9.2 Register Context Source

5.9.2.1 Description

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

5.9.2.2 Use case diagram

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

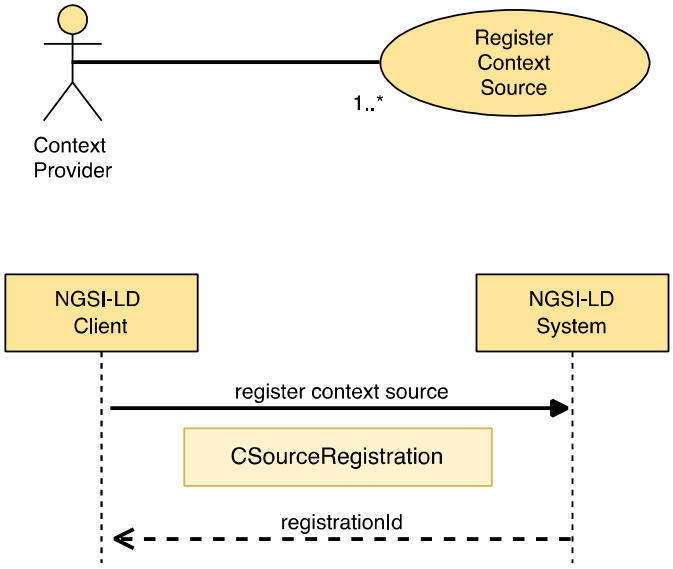

Figure 5.9.2.2-1: Register context source use case

5.9.2.3 Input data

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

5.9.2.4 Behaviour

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

- If the data types and restrictions expressed by clause 5.2.9 are not met by the Context Source Registration, then an error of type *BadRequestData* shall be raised.
- If a "contextSourceInfo" array is defined and the restrictions expressed by clause 4.3.3.6 are not met by the Context Source Registration, then an error of type *BadRequestData* shall be raised.
- If the Context Source to be registered has its *mode* property defined as **exclusive**, the following additional restrictions apply:
  - If an **exclusive** or **redirect** Context Source Registration already matches against the Entity *id* (URI) and any of the Attributes defined in the registration, an error of type *Conflict* shall be raised.
  - If an Entity already exists for the supplied Entity *id* (URI) and the existing Entity contains any of the Attributes defined in the registration, an error of type *Conflict* shall be raised.
- If the Context Source to be registered has its *mode* property defined as **redirect**, the following additional restriction applies:
  - If an existing Entity already matches the Context Source Registration, an error of type *Conflict* shall be raised.
- If the Context Source to be registered has its *mode* property defined as **auxiliary**, the following additional restriction applies:
  - If the *operations* property is not defined as one of: "retrieveOps", "retrieveEntity" or "queryEntity", an error of type *BadRequestData* shall be raised.
- If the property *expiresAt* is not defined then the Context Source Registration shall last forever (or until it is deleted from the system).
- If *expiresAt* is a date and time in the past, an error of type *BadRequestData* shall be raised.
- If *expiresAt* 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.

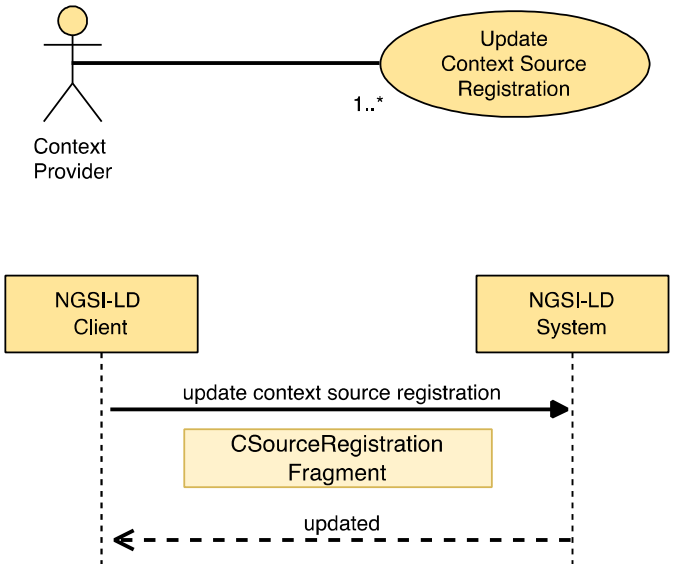

Figure 5.9.3.2-1: Update context source registration use case

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 a "contextSourceInfo" array is defined and the restrictions expressed by clause 4.3.3.6 are not met by the Context Source Registration, 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 in 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.
- If the Context Source Registration to be updated has its *mode* property defined as **exclusive**, the following additional restrictions apply:
  - If an **exclusive** or **redirect** Context Source Registration already matches against the Entity *id* (URI) and any of the Attributes defined in the registration, an error of type *Conflict* shall be raised.
  - If an Entity already exists for the supplied Entity *id* (URI) and the existing Entity contains any of the Attributes defined in the registration, an error of type *Conflict* shall be raised.

- If the Context Source Registration to be updated has its *mode* property defined as **redirect**, the following additional restriction applies:
  - If an existing Entity already matches the Context Source Registration, an error of type *Conflict* shall be raised.
- If the Context Source to be updated has its *mode* property defined as **auxiliary**, the following additional restriction applies:
  - If the *operations* property is not defined as one of: "retrieveOps", "retrieveEntity" or "queryEntity", an error of type *BadRequestData* shall be raised.
- 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.

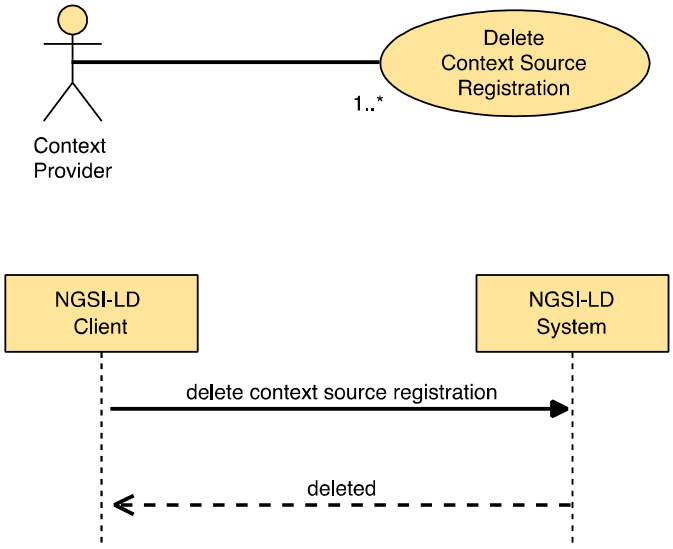

Figure 5.9.4.2-1: Delete context source registration use case

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 endpoint does not know about the target context source registration, because there is no existing context source registration whose id (URI) is equivalent, then an error of type *ResourceNotFound* shall be raised.
- Otherwise the context source registration shall be removed.

5.9.4.5 Output data

None.

5.10 Context Source Discovery

5.10.1 Retrieve Context Source Registration

5.10.1.1 Description

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

5.10.1.2 Use case diagram

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

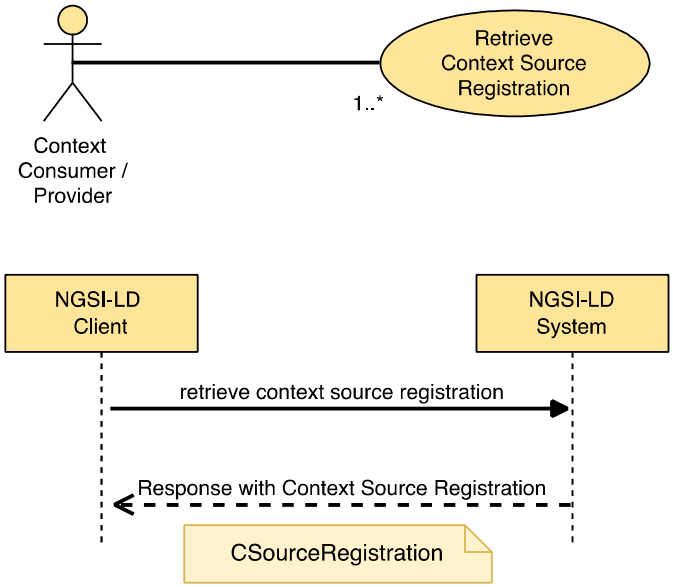

Figure 5.10.1.2-1: Retrieve context source registration use case

5.10.1.3 Input data

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

5.10.1.4 Behaviour

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

- Term to URI expansion of Attribute names shall be observed as mandated by clause 5.5.7.
- Otherwise return a JSON-LD object representing the Context Source Registration as mandated by clause 5.2.9.

5.10.1.5 Output data

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

5.10.2 Query Context Source Registrations

5.10.2.1 Description

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

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

5.10.2.2 Use case diagram

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

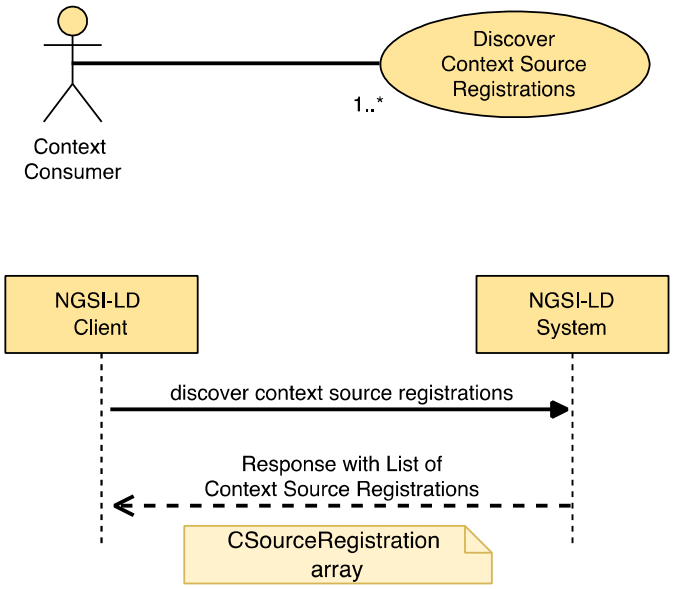

Figure 5.10.2.2-1: Discover context source registrations use case

5.10.2.3 Input data

- A reference to a JSON-LD @context (optional).
- A selector of Entity types as specified by clause 4.17. Both type name (short hand string) and fully qualified type name (URI) are allowed (optional).
- A list (one or more) of Entity identifiers (optional).
- A list (one or more) of Attribute names (called query projection attributes) (optional).

- An id pattern as a regular expression (optional).
- An NGSI-LD query (to filter out Entities by Attribute values, used here to identify relevant attributes) as per clause 4.9 (optional).
- An NGSI-LD geoquery (to filter out Entities by spatial relationships, used here to identify relevant GeoProperties and for geographical scoping) as per clause 4.10 (optional).
- In the case of GeoJSON representation:
  - The name of the **GeoProperty** attribute to use as the geometry for the GeoJSON representation as mandated by clause 4.5.16 (optional).
  - A datasetId specifying which instance of the value is to be selected if the **GeoProperty** value has multiple instances as defined by clause 4.5.5 (optional).
- An NGSI-LD temporal query as per clause 4.11 (optional).
- An NGSI-LD context source query as per clause 4.9 (optional).
- A NGSI-LD Scope query as mandated by clause 4.19 (optional).
- A limit to the number of Context Source Registrations to be retrieved. See clause 5.5.9.
- A specified language filter as per clause 4.15 (optional).

It is not possible to retrieve a set of context source registrations related to entities by only specifying desired entity identifiers, without further specifying restrictions on the entities' types or attributes, either explicitly, via lists of Entity types or of Attribute names, or implicitly, within an NGSI-LD query or geoquery.

5.10.2.4 Behaviour

- Execute the behaviour defined in clause 5.5.4 on JSON-LD validation.
- At least one of the following input data shall be provided:

  a) *selector of Entity Types;*

  b) *list of Attribute names*;

  c) *NGSI-LD query*;

  d) *NGSI-LD geoquery*.

  If none of them is provided, then an error of type *BadRequestData* shall be raised (too wide query). Attributes specified in *NGSI-LD query* or *NGSI-LD geoquery* shall be used for matching RegistrationInfo elements in the same way as the attributes in the *list of Attribute names*.
- If the list of Entity identifiers includes a URI which it is not valid, or the query, geoquery 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.
- Term to URI expansion of type and Attribute names shall be performed, as mandated by clause 5.5.7.
- Otherwise, implementations shall run a query that shall return context source registrations that meet **all** the applicable conditions:
  - If present, the entity specification in the query consisting of a combination of entity type selector and entity id/entity id pattern (optional) matches an *EntityInfo* specified in a *RegistrationInfo* of the information property in a context source registration. If there is no *EntityInfo* specified in the *RegistrationInfo*, the entity specification is considered matching. This matching is further described in clause 5.12.

- If present, at least one Attribute name specified in the query matches one Property or Relationship in the *RegistrationInfo* element of the information property in a context source registration.. If no Properties or Relationships are specified in the *RegistrationInfo*, the Attribute names are considered matching. This matching is further described in clause 5.12.
- If present, the geoquery is matched against the *GeoProperty* identified in the geoquery. If no *GeoProperty* is specified in the geoquery, the default property is 'location'. The geoquery matches the *GeoProperty* specified in the Context Source Registration, if the location directly matches or if the location possibly contains locations that would match the geoquery.
- If no temporal query is present, only Context Source Registrations for Context Sources providing latest information, i.e. without specified time intervals, are considered.
- If a temporal query is present, only Context Source Registrations with specified time intervals, i.e. *observationInterval* or *managementInterval* are considered. If the *timeproperty* is *observedAt* or no *timeproperty* is specified in the temporal query (default: *observedAt*), the temporal query is matched against the *observationInterval* (if present). If the *timeproperty* is *createdAt*, *modifiedAt* or *deletedAt*, the temporal query is matched against the *managementInterval* (if present). If the relevant interval is not present, there is no match:
  - The semantics of the match is that the "timeAt" in the case of the "before" and "after" relation is contained in or is an endpoint of a time period included in the specified time interval. In the case of the "between" relation there is a match if there is an overlap between the interval specified by the "timeAt" and "endTimeAt" and the specified time interval.
- If present, the conditions specified by the context source query filter match the respective Context Source Properties (as mandated by clause 4.9).
- If present, the Scope query (as mandated by clause 4.19) is matched against the scope property.

- Pagination logic shall be in place as mandated by clause 5.5.9.

5.10.2.5 Output data

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

5.11 Context Source Registration Subscription

5.11.1 Introduction

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

5.11.2 Create Context Source Registration Subscription

5.11.2.1 Description

This operation allows creating a new Context Source Registration Subscription.

5.11.2.2 Use case diagram

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

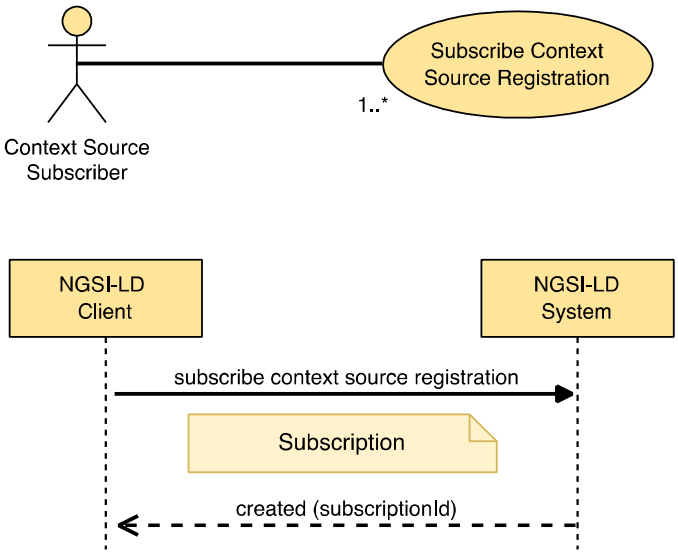

Figure 5.11.2.2-1: Subscribe Context Source Registration use case

5.11.2.3 Input data

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

5.11.2.4 Behaviour

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

- If a temporal query in the *temporalQ* element is present, only Context Source Registrations with specified time intervals are considered. If the *timeproperty* is *observedAt* or no *timeproperty* is specified in the temporal query (default: *observedAt*), the temporal query is matched against the *observationInterval* (if present). If the *timeproperty* is *createdAt*, *modifiedAt* or *deletedAt*, the temporal query is matched against the *managementInterval* (if present). If the relevant interval is not present, there is no match:
  - The semantics of the match is that the "timeAt" in the case of the "before" and "after" relation is contained in or is an endpoint of a time period included in the specified time interval. In the case of the "between" relation there is a match if there is an overlap between the interval specified by the "timeAt" and "endTimeAt" and the specified time interval.
- If present, the conditions specified by the context source filter match the respective Context Source Properties (as mandated by clause 4.9).

- 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.
- If present, the conditions specified by the context source query match the respective Context Source Properties (as mandated by clause 4.9).
- If present, the Scope query (as mandated by clause 4.19) is matched against the *scope* property.

5.11.2.5 Output data

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

5.11.3 Update Context Source Registration Subscription

5.11.3.1 Description

This operation allows updating an existing Context Source Registration Subscription.

5.11.3.2 Use case diagram

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

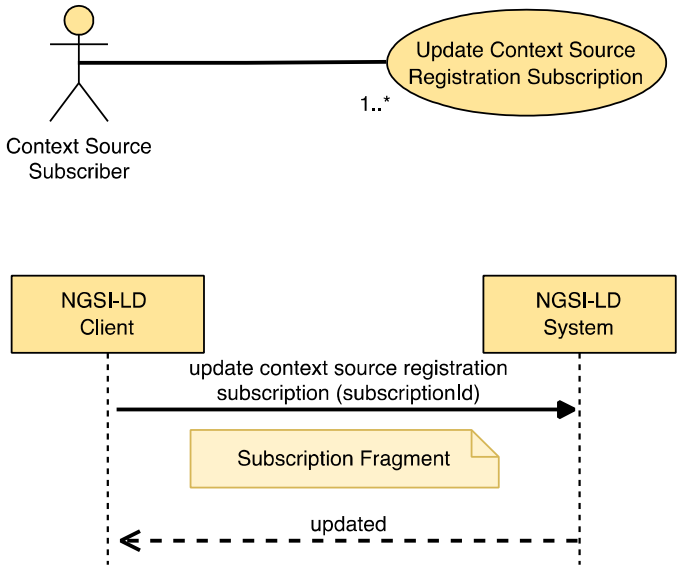

Figure 5.11.3.2-1: Update Context Source Registration Subscription use case

5.11.3.3 Input data

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

5.11.3.4 Behaviour

- If the Subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the data types and restrictions expressed by clause 5.2.12 are not met by the Subscription Fragment, then an error of type *BadRequestData* shall be raised.
- Then, implementations shall modify the target subscription as mandated by clause 5.5.8.
- Finally, send a notification with all currently matching Context Source Registrations.

5.11.3.5 Output data

None.

5.11.4 Retrieve Context Source Registration Subscription

5.11.4.1 Description

This operation allows retrieving an existing Context Source Registration Subscription.

5.11.4.2 Use case diagram

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

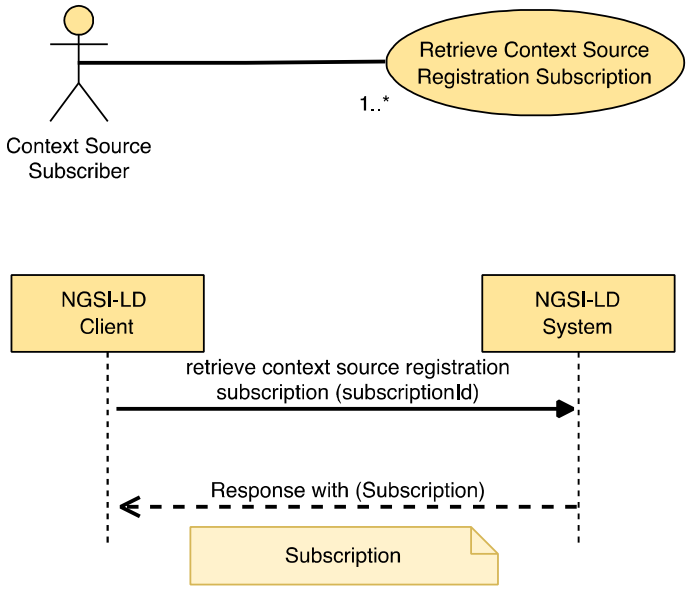

Figure 5.11.4.2-1: Retrieve Context Source Registration Subscription use case

5.11.4.3 Input data

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

5.11.4.4 Behaviour

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

5.11.4.5 Output data

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

5.11.5 Query Context Source Registration Subscriptions

5.11.5.1 Description

This operation allows querying existing Context Source Registration Subscriptions.

5.11.5.2 Use case diagram

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

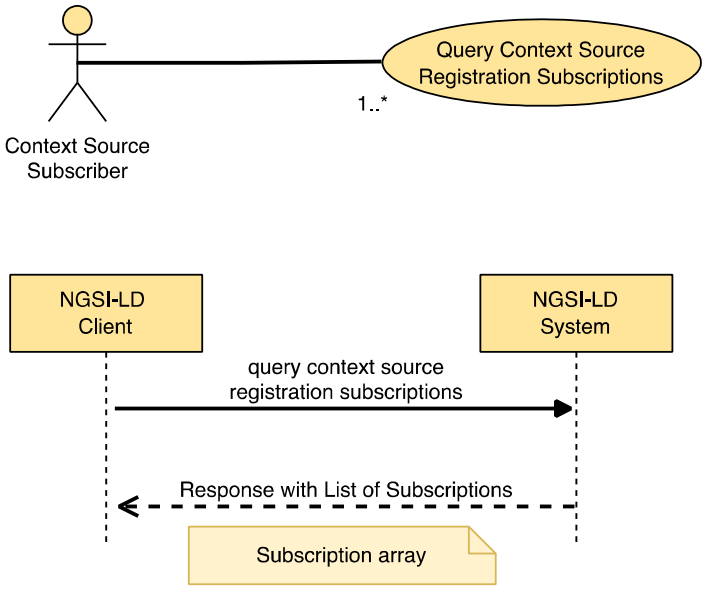

Figure 5.11.5.2-1: Query Context Source Registration Subscriptions use case

5.11.5.3 Input data

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

5.11.5.4 Behaviour

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

5.11.5.5 Output data

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

5.11.6 Delete Context Source Registration Subscription

5.11.6.1 Description

This operation allows deleting an existing Context Source Registration Subscription.

5.11.6.2 Use case diagram

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

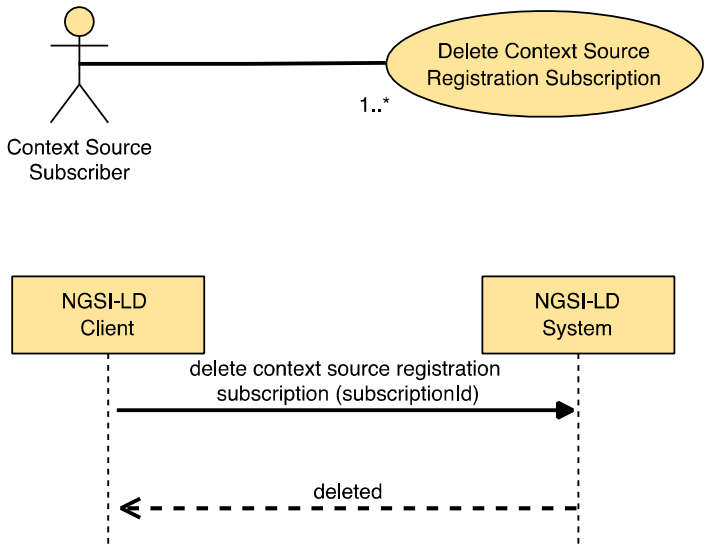

Figure 5.11.6.2-1: Delete Context Source Registration Subscriptions use case

5.11.6.3 Input data

- A subscription identifier (URI).

5.11.6.4 Behaviour

- If the subscription Id is not present or it is not a valid URI, then an error of type *BadRequestData* shall be raised.
- If the subscription id provided does not correspond to any existing subscription in the system then an error of type *ResourceNotFound* shall be raised.
- Otherwise implementations shall delete the Context Source Registration Subscription and no longer perform notifications concerning that 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 geoquery. If either the watched Attributes or the Attributes in the notification are not present or of length 0, all possible Attributes (if present in the Context Source Registrations) for fitting entities match.
- If a subscription does not define a "timeInterval" term, the csource notification shall be sent on initial subscription and whenever there is a change in a matching csource registration. Such a change may be triggered by the creation of a new matching csource registration, the update of a csource registration (whether matching before the update, after the update or in both cases) or the deletion of a matching csource registration. The notification message shall include the matching csource registration(s) together with the appropriate trigger reason in the "triggerReason" member.

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

- A csource notification shall be sent as follows:

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

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

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

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

  - If the notification is sent successfully:

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

  - If the notification is not sent successfully:

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

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

5.12 Matching Context Source Registrations

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

The relevant specification information in the query for Context Source Registrations are the selector of Entity Types (if present), the list of Entity identifiers (if present), the id pattern (if present) and the list of Attribute names (if present). In the case of subscriptions to context source registrations, it is the Entities as specified in the array of type *EntitySelector* 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 way relevant Entities are specified differs in queries and subscriptions, they consist of the same information, so for the purpose of this clause, the specification of Entity Types or Attributes refers to the relevant elements for matching, i.e. Entity Types, Entity identifiers, id pattern and Attribute names. A specification of Entity Types or Attributes shall contain at least one of:

a) selector of Entity Types; or

b) list of Attribute names.

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

- If present, the selector of Entity Types, Entity identifiers and id pattern match at least one of the *EntityInfo* elements of the *RegistrationInfo* (see below).

- If present, the Attribute identifiers match the combination of Properties and Relationships specified in the *RegistrationInfo* (see below).

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

- The *EntityInfo* contains neither an *id* nor an *idPattern*.

- One of the specified entity identifiers matches the *id* in the *EntityInfo*.
- At least one of the specified entity identifiers matches the *idPattern* in the *EntityInfo*.
- The specified id pattern matches the *id* in the *EntityInfo*.
- Both a specified id pattern and an *idPattern* in the *Entity Info* are present (since in the general case it is not easily feasible to determine if there can be identifiers matching both patterns).

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

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

5.13 Storing, Managing and Serving @contexts

5.13.1 Introduction

NGSI-LD Brokers optionally (see clause 4.3.5) offer the capability to store and serve @contexts to clients. The stored @contexts may be managed by clients directly, via the APIs specified in clause 5.13. Clients can store custom user @contexts at the Broker, effectively using the Broker as a @context server.

Moreover, in order to optimize performance, NGSI-LD Brokers may automatically store and use the stored copies of common @contexts as a local cache, downloading them just once, thus avoiding fetching them over and over again at each NGSI-LD request. In order for the Broker to understand if a needed @context is already in the local storage or not, the Broker uses the URL, where the @context is originally hosted, as an identifier for it in the local storage. Consequently, the Broker has no ability to cache @contexts that arrive to it as **embedded** parts within the NGSI-LD documents, since they are not uniquely (and implicitly) identified by any URL; Brokers only cache @contexts that are referred to by means of explicit URLs (either in the HTTP Link header or as URLs in the payload body). Thus, the **recommended best-practice, in order to exploit caching, is that clients do not embed their user @contexts into their NGSI-LD documents**; instead clients should explicitly host their user @contexts at their premises, or use the Broker's capability to host user @contexts on their behalf.

When an external @context is stored, either explicitly upon a client's request or implicitly downloaded for caching purposes, the NGSI-LD Broker generates a unique local @context identifier. The original @context's URL, if any, is stored alongside the generated local id. The local id is then used for subsequent managing operations on the specific @context, that are specified in clauses 5.13.2 to 5.13.5. Moreover, the Broker tags the entry with the current timestamp, so that, subsequently, clients can check the timestamp before deciding whether to force a refresh of the stored copy of the @context. This is primarily intended as a means for clients to well-behave, thus avoiding triggering continuous refresh of a stored @context on the Broker, for fear that it is not at the latest version.

Stored @contexts are flagged as one of three kinds: "Cached", "Hosted", "ImplicitlyCreated".

- **Cached:**
  @contexts implicitly and automatically fetched by the Broker from external URLs during normal NGSI-LD operations are flagged as "Cached". A locally unique identifier is generated for each @context not already in the internal storage. The downloaded content, its URL and the current time in UTC are stored alongside the locally unique identifier. Implementations shall periodically invalidate the "Cached" @contexts. Depending on the binding of the NGSI-LD API to a specific protocol, that specific protocol may provide explicit indications about expiration times of cached content. In such cases, implementations shall comply with the indications provided by the protocol. Implementations should assign a heuristic expiration time when an explicit time is not specified. **Entries flagged as "Cached" shall not be served by NGSI-LD Brokers on-demand, but only be used as a local cache to improve performance**.
- **Hosted:**
  @contexts that are explicitly added by users are flagged as "Hosted". These entries shall be served by NGSI-LD Brokers on-demand.

- **ImplicitlyCreated:**
  @contexts that are implicitly, but *ex-novo*, created by the Broker as a result of a user request are flagged as "ImplicitlyCreated". For instance, when a client creates a subscription using an @context that is an array, and the Broker has to notify with Content-Type application/json, then the Broker needs this @context array to be hosted at a URL. Hence the Broker has to create a new @context that is an array, and it is going to be served from an own URL. These entries shall be served by NGSI-LD Brokers on-demand.

5.13.2 Add @context

5.13.2.1 Description

With this operation, a client can ask the Broker to store the full content of a specific @context, by giving it to the Broker.

5.13.2.2 Use case diagram

A client can add an @context to be stored within an NGSI-LD system as shown in figure 5.13.2.2-1.

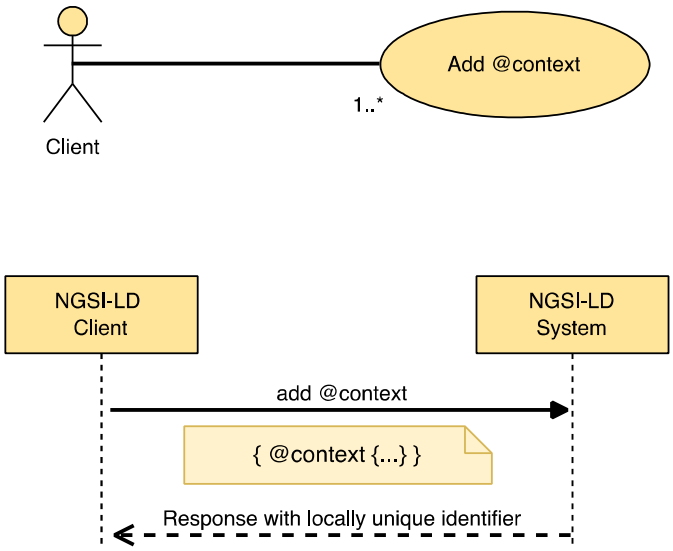

Figure 5.13.2.2-1: Add @context use case

5.13.2.3 Input data

A JSON object that has a top-level field named @context, i.e. a JSON object representing a JSON-LD "local context". As specified in the JSON-LD specification [2], all extra information located outside of the @context subtree in the referenced object shall be discarded.

5.13.2.4 Behaviour

A new entry is created in the local storage and a locally unique identifier is generated for it. The JSON object representing the client-supplied @context and the current UTC time are stored alongside the locally unique identifier. That identifier shall be given back as a result in the output data. The entry is flagged as being of kind "Hosted".

The behaviour described in clause 5.5.4 about JSON and JSON-LD validation shall be applied in case of invalid @context.

5.13.2.5 Output data

The locally unique identifier that identifies the @context in the Broker's internal storage.

5.13.3 List @contexts

5.13.3.1 Description

With this operation a client can obtain a list of URLs that represent all of the @contexts stored in the local context store of the Broker. Each URL can be used to download the corresponding @context, and, in case the @context's kind is "Cached", it shall be the original URL the Broker downloaded the @context from.

In case a "details" flag is set to *true*, the client obtains a list of JSON objects, each representing information (metadata) about an @context currently stored by the Broker. Each JSON object contains information about the @context's original URL (if any), its local identifier in the Broker's storage, its kind ("Cached", "Hosted" and "ImplicitlyCreated"), its creation timestamp, its expiry date (if "Cached"), and additional optional information.

5.13.3.2 Use case diagram

A client can list all @contexts stored within an NGSI-LD system as shown in figure 5.13.3.2-1.

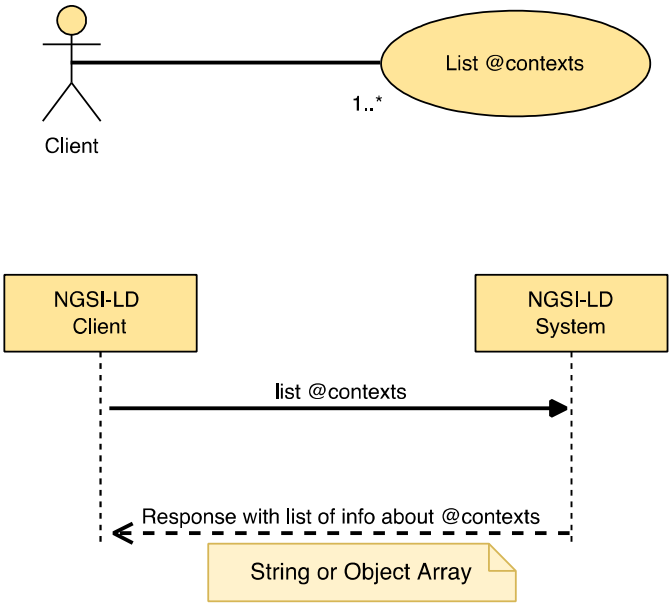

Figure 5.13.3.2-1: List @contexts use case

5.13.3.3 Input data

- A *kind* filter indicating the kind of stored @contexts that are to be included in the output list. Currently, possible kinds are "Cached", "Hosted" and "ImplicitlyCreated" (optional).
- A boolean *details* flag indicating that detailed JSON objects representing metadata about the stored @contexts instead of simple URLs are requested (optional).

5.13.3.4 Behaviour

The Broker shall provide a URL or JSON object for each @context currently stored in the internal Broker's storage, that match the filter. If no filter is specified, all kinds are included.

5.13.3.5 Output data

A list of URLs, or a list of resulting JSON objects containing the following fields:

- URL;
- localId;

- kind;
- timestamp;
- lastUsage [OPTIONAL];
- numberOfHits [OPTIONAL];
- extraInfo [OPTIONAL, used by implementations to report any kind of custom information].

5.13.4 Serve @context

5.13.4.1 Description

With this operation a client can obtain the full content of a specific @context (only for @contexts of kind "Hosted" or "ImplicitlyCreated"), which is currently stored in the Broker's internal storage, or its metadata (for all kinds of stored @contexts).

5.13.4.2 Use case diagram

A client can request the Broker to serve a specific @context stored within the NGSI-LD system as shown in figure 5.13.4.2-1.

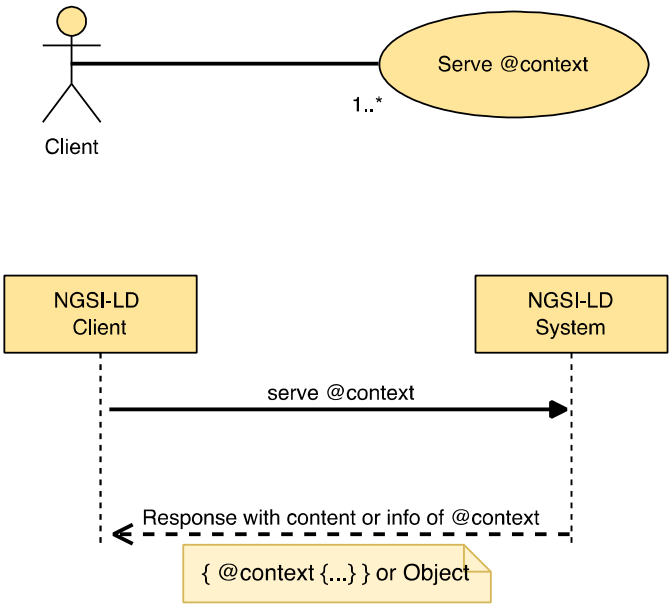

Figure 5.13.4.2-1: Serve @context use case

5.13.4.3 Input data

- The locally unique identifier that identifies the desired @context in the Broker's internal storage. Such unique identifiers are obtained by the client as a result of either a "Add @context" (clause 5.13.2) API operation or of a "List @contexts" (clause 5.13.3) API operation. For @contexts of kind "Cached" this can also be the original URL the Broker downloaded the @context from.
- A boolean *details* flag indicating that a JSON object representing metadata about the @context, instead of the full content, is requested (optional).

5.13.4.4 Behaviour

- If *details* is set to false, or details is not present, the Broker shall give back the full content of the @context that corresponds to the indicated local identifier, serving it from its internal storage, if the @context that corresponds to the indicated local identifier is of kind "Hosted" or "ImplicitlyGenerated". It shall give back *OperationNotSupported* error if it is of kind "Cached". It shall give back *ResourceNotFound* if the identifier is not found.
- Otherwise, if *details* is set to true, the Broker shall give back metadata about the @context that corresponds to the indicated local identifier. It shall give back *ResourceNotFound* error if the identifier is not found.

5.13.4.5 Output data

The full content of the indicated @context (or its metadata as specified in clause 5.13.3.5), or *ResourceNotFound/OperationNotSupported* errors.

5.13.5 Delete and Reload @context

5.13.5.1 Description

With this operation, a client supplies a local identifier to the Broker, indicating a stored @context, that the Broker shall remove from its storage. For @contexts of kind "Cached" this can also be the original URL the Broker downloaded the @context from. If the entry in the local storage that corresponds to the identifier is itself an array of @contexts, this operation will **not** delete the children, i.e. the @contexts in the array, but just the entry.

5.13.5.2 Use case diagram

A client can request the Broker to delete (and optionally reload) a specific @context stored within the NGSI-LD system as shown in figure 5.13.5.2-1.

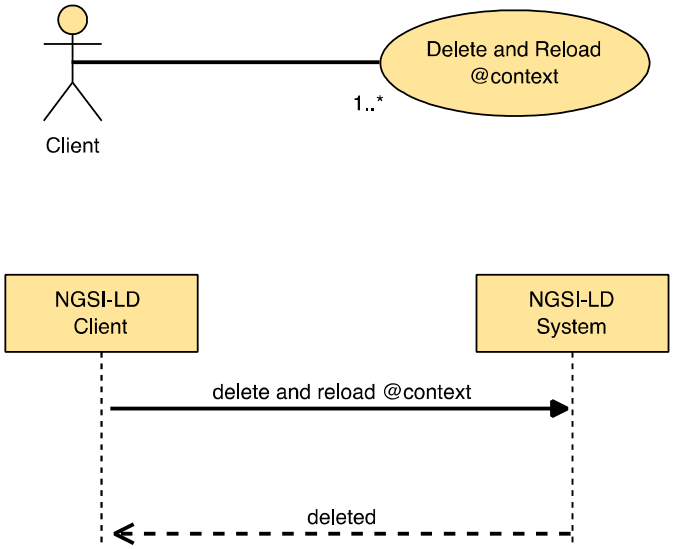

Figure 5.13.5.2-1: Delete and Reload @context use case

5.13.5.3 Input data

- The locally unique identifier that identifies the desired @context in the Broker's internal storage. For @contexts of kind "Cached" this can also be the original URL the Broker downloaded the @context from.
- A *reload* boolean flag indicating that reloading of the @context shall be attempted (optional).

5.13.5.4 Behaviour

- If the @context identifier is not supplied, then an error of type *BadRequestData* shall be raised.
- If the @context identifier does not correspond to any existing entry in the @context storage, then an error of type *ResourceNotFound* shall be raised.
- If *reload* is true and the kind of the @context is "Cached", implementations shall try to re-download the identified @context from its original URL, before removing it from the internal storage. If downloading fails, or the downloaded @context is invalid according to JSON and JSON-LD validation of clause 5.5.4, then an error of type *LdContextNotAvailable* shall be raised. More detailed information about the errors shall be specified in the ProblemDetails (see IETF RFC 7807 [10]) field of the response. In case of any error, the operation ends without removing the existing @context. Otherwise, the existing @context is replaced with the newly downloaded one.
- If *reload* is true and the kind of the @context is **not** "Cached", implementations shall return a *BadRequestData* error.
- If *reload* is false (or *reload* is not supplied), implementations shall remove from the internal storage the @context that corresponds to the given identifier. The local identifier is used for finding the @contexts in the internal Broker's storage. If the local identifier is not in the storage, a *ResourceNotFound* error shall be raised.

5.13.5.5 Output data

Void.

6 API HTTP Binding

6.1 Introduction

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

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

6.2 Global Definitions and Resource Structure

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

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

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

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

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

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

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

All resource URIs 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.

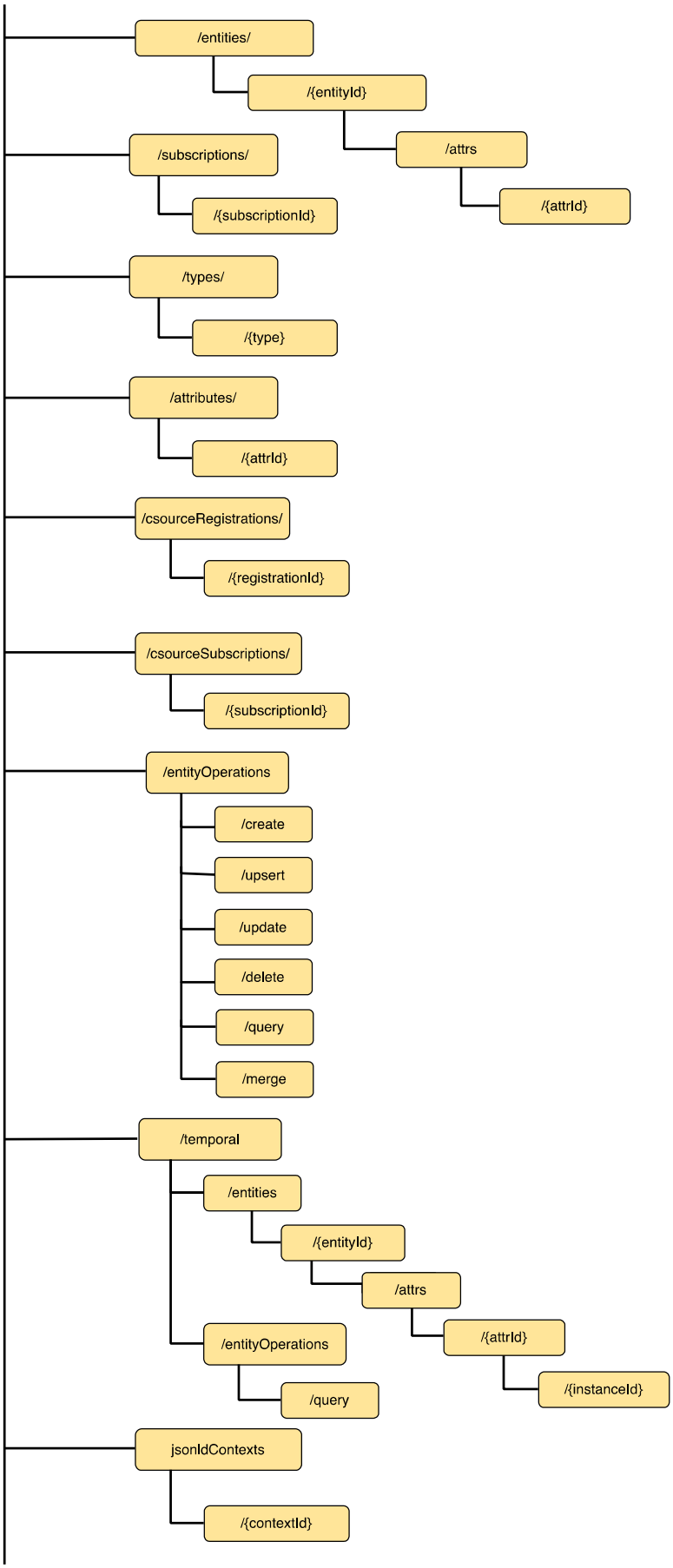

Figure 6.2-1: Resource URI structure of the NGSI-LD API

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

Resource Name	Resource URI	HTTP Method	Meaning	Clauses
Entity List	/entities/	POST	Entity creation	5.6.1; 6.4.3.1
		GET	Query entities	5.7.2; 6.4.3.2
Entity by id	/entities/{entityId}	GET	Entity retrieval by id	5.7.1; 6.5.3.1
		DELETE	Entity deletion by id	5.6.6; 6.5.3.2
		PATCH	Entity merge by id	5.6.17; 6.5.3.4
		PUT	Entity replacement by id	5.6.18; 6.5.3.3
Attribute List	/entities/{entityId}/attrs/	POST	Append Attributes to Entity	5.6.3; 6.6.3.1
		PATCH	Update Attributes of an Entity	5.6.2; 6.6.3.2
Attribute by id	/entities/{entityId}/attrs/{attrId}	PATCH	Partial Attribute update	5.6.4; 6.7.3.1
		DELETE	Attribute delete	5.6.5; 6.7.3.2
		PUT	Attribute replace	5.6.19; 6.7.3.3
Subscriptions List	/subscriptions/	POST	Create Subscription	5.8.1; 6.10.3.1
		GET	Retrieve list of Subscriptions	5.8.4; 6.10.3.2
Subscription by Id	/subscriptions/{subscriptionId}	GET	Subscription retrieval by id	5.8.3; 6.11.3.1
		PATCH	Subscription update by id	5.8.2; 6.11.3.2
		DELETE	Subscription deletion by id	5.8.5; 6.11.3.3
Entity Types	/types/	GET	Retrieve available entity types	5.7.5 and 5.7.6; 6.25.3.1
Entity Type	/types/{type}	GET	Details about available entity type	5.7.7; 6.26.3.1
Attributes	/attributes/	GET	Available attributes	5.7.8 and 5.7.9; 6.27.3.1
Attribute	/attributes/{attrId}	GET	Details about available attribute	5.7.10; 6.28.3.1
Context source registration list	/csourceRegistrations/	POST	CSource registration creation	5.9.2; 6.8.3.1
		GET	Discover CSource registrations	5.10.2; 6.8.3.2
Context source registration by Id	/csourceRegistrations/{registrationId}	GET	CSource registration retrieval by id	5.10.1; 6.9.3.1
		PATCH	CSource registration update by id	5.9.3; 6.9.3.2
		DELETE	CSource registration deletion by id	5.9.4; 6.9.3.3
Context source registration subscription list	/csourceSubscriptions/	POST	Create subscription to CSource registration	5.11.2; 6.12.3.1
		GET	Retrieval of list of subscription to CSource registration	5.11.5; 6.12.3.2
Context source registration subscription by Id	/csourceSubscriptions/{subscriptionId}	GET	CSource registration subscription retrieval by id	5.11.4; 6.13.3.1
		PATCH	CSource registration subscription update by id	5.11.3; 6.13.3.2
		DELETE	CSource registration subscription deletion by id	5.11.6; 6.13.3.3
Entity Operations. Create	/entityOperations/create	POST	Batch Entity creation	5.6.7; 6.14.3.1
Entity Operations. *Upsert*	/entityOperations/upsert	POST	Batch Entity create or update (*upsert*)	5.6.8; 6.15.3.1
Entity Operations. Update	/entityOperations/update	POST	Batch Entity update	5.6.9; 6.16.3.1
Entity Operations. Delete	/entityOperations/delete	POST	Batch Entity deletion	5.6.10; 6.17.3.1
Entity Operations. Query	/entityOperations/query	POST	Query entities based on POST	5.7.2; 6.23.3.1
Entity Operations. Merge	/entityOperations/merge	POST	Batch Entity merge	5.6.20; 6.31.3.1
Entity Temporal Evolution	/temporal/entities/	POST	Temporal Representation of Entity creation	5.6.11; 6.18.3.1
		GET	Query temporal evolution of Entities	5.7.4; 6.18.3.2
Temporal Representation of Entity by id	/temporal/entities/{entityId}	GET	Temporal Representation of Entity retrieval by id	5.7.3; 6.19.3.1
		DELETE	Temporal Representation of Entity deletion by id	5.6.16; 6.18.3.2

Resource Name	Resource URI	HTTP Method	Meaning	Clauses
Temporal Representation of Attribute List	/temporal/entities/{entityId}/attrs/	POST	Temporal Representation of Attribute instance addition	5.6.12; 6.20.3.1
Temporal Representation of Attribute by id	/temporal/entities/{entityId}/attrs/{attrId}	DELETE	Attribute from Temporal Representation of Entity deletion	5.6.13; 6.21.3.1
Temporal Representation of Attribute Instance by id	/temporal/entities/{entityId}/attrs/{attrId}/{instanceId}	PATCH	Attribute Instance update	5.6.14; 6.22.3.1
		DELETE	Attribute Instance deletion by instance id	5.6.15; 6.22.3.2
Temporal Query Operation	/temporal/entityOperations/query	POST	Temporal Representation of Entity Query based on POST	5.7.4; 6.24.3.1
Add and List @context	/jsonldContexts	POST	Add a user @context to the internal cache	5.13.2; 6.29.3.1
		GET	List all cached @contexts	5.13.3; 6.29.3.2
Serve, Delete and Reload @context	/jsonldContexts/{contextId}	GET	Serve one specific user @context	5.13.4; 6.30.3.1
		DELETE	Delete one specific @context from internal cache, possibly re-inserting a freshly downloaded copy of it	5.13.5; 6.30.3.2

6.3 Common Behaviours

6.3.1 Introduction

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

6.3.2 Error Types

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

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

Error Type	HTTP status
https://uri.etsi.org/ngsi-ld/errors/InvalidRequest	400
https://uri.etsi.org/ngsi-ld/errors/BadRequestData	400
https://uri.etsi.org/ngsi-ld/errors/AlreadyExists	409
https://uri.etsi.org/ngsi-ld/errors/OperationNotSupported	422
https://uri.etsi.org/ngsi-ld/errors/ResourceNotFound	404
https://uri.etsi.org/ngsi-ld/errors/InternalError	500
https://uri.etsi.org/ngsi-ld/errors/TooComplexQuery	403
https://uri.etsi.org/ngsi-ld/errors/TooManyResults	403
https://uri.etsi.org/ngsi-ld/errors/LdContextNotAvailable	503
https://uri.etsi.org/ngsi-ld/errors/NoMultiTenantSupport	501
https://uri.etsi.org/ngsi-ld/errors/NonexistentTenant	404

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

- "Method Not Allowed" (405) which shall be raised when a client invokes a wrong HTTP verb over a resource. Implementations shall provide the allowed HTTP methods as mandated by IETF RFC 7231 [3] in section 6.5.5.
- "Request Entity too large" (413) which shall be raised when the HTTP input data stream provided by a client was too large i.e. too many bytes.

- "Length required" (411) which shall be raised when an HTTP request provided by a client does not define the "Content-Length" HTTP header.
- "Unsupported Media Type" (415) which shall be raised when an HTTP request payload body (as per the "Content-Type" header) it is not "application/json" nor "application/ld+json".
- "Not Acceptable" (406) which shall be raised when the response media types that are acceptable by a client (as per the "Accept" header) do not include or expand to "application/json" nor "application/ld+json".

6.3.3 Reporting errors

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

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

6.3.4 HTTP request preconditions

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

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

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

For GET HTTP requests implementations shall check the following preconditions:

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

The order of the list above is significant. If the Accept header can be expanded to more than one of the options of the list, the first one of the list shall be selected, unless amended by the HTTP Accept header processing rules, e.g. the presence of a "q" parameter indicating a relative weight, (as mandated by IETF RFC 7231 [3], section 5.3.2) require otherwise.

If the Accept header is not present, "application/json" shall be assumed.

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

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

Notwithstanding the above, if the Accept Header is set to "application/geo+json":

- For Context Information Consumption operations only, specifically "Retrieve Entity" (see clause 5.7.1) and "Query Entity" (clause 5.7.2) GeoJSON is considered as an acceptable content type and a GeoJSON payload will be returned.

- For all other operations, the request will result in a Not Acceptable Media Type error, returning a **406** HTTP status code and the body of the message shall contain the list of the available representations of the resources.

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], section 6.2. 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, PATCH or PUT and the Content-Type header is "application/json", then the @context shall be obtained from a Link Header as mandated by JSON-LD [2], section 6.2. In the absence of such Link Header, then the "@context" shall be the default @context. In any case, if the request payload body (as JSON) contains a "@context" term, then an HTTP error response of type *BadRequestData* shall be raised.

- If the request verb is POST, PATCH or PUT and the Content-Type header is "application/ld+json", then the associated @context shall be obtained from the request payload body itself. If no @context can be obtained from the request payload body, then an HTTP error response of type *BadRequestData* shall be raised. In any case, the presence of a JSON-LD Link header in the incoming HTTP request when the Content-Type header is "application/ld+json" shall result in an HTTP error response of type *BadRequestData*.

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

In contrast, GET and DELETE operations always take their input @context from the JSON-LD Link Header.

6.3.6 HTTP response common requirements

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

- If the target response's MIME type is "application/json" such response shall include a Link to the associated JSON-LD @context as mandated by [2], section 6.2.

- If the target response's MIME type is "application/ld+json", then the response payload body provided by the HTTP response shall include a JSON-LD @context.

- If the target response's MIME type is "application/geo+json" and the Prefer Header [26] is omitted or set to "body=ld+json", then the response payload body provided by the HTTP response shall include a JSON-LD @context, and the representation of the entities shall be in GeoJSON format in the response payload body.

- If the target response's MIME type is "application/geo+json" and the Prefer Header [26] is set to "body=json" such response shall include a Link to the associated JSON-LD @context as mandated by [2], section 6.2 and the representation of the entities shall be in GeoJSON format in the response payload body, and "@context" shall be omitted from the payload body.

Operations that result in an error that return a payload shall always respond with MIME type "application/json", regardless of the Accept header. It is assumed that if a client application understands any of the supported MIME types, the application shall understand "application/json" errors.

Operations where the response payload body is not present such as successful POST, or PATCH or PUT operations and all error responses, do not include the Link header in the response. Only Fully Qualified Names shall be used in the payload body of error responses, as there is no context present.

No Content-Length HTTP header shall be present if the response code is 204.

6.3.7 Representation of Entities

For HTTP GET operations performed over the resource/entities and all of its sub-resources, Context Broker implementations shall support the parameter specified in table 6.3.7-1, which specifies all possible supported representations formats.

In contrast, at a minimum, registered Context Source implementations shall support the normalized representation of Entities as default. When a registered Context Source is unable to support additional representations, a 501 Not Implemented Error shall be raised.

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 "normalized", a normalized representation of Entities shall be provided as defined by clause 4.5.1, with Attributes returned in the normalized representation as defined in clauses 4.5.2.2, 4.5.3.2, 4.5.18.2 and 4.5.20.2. When its value includes the keyword "concise", a concise lossless representation of Entities shall be provided as defined by clause 4.5.1. with Attributes returned in the concise representation as defined in clauses 4.5.2.3, 4.5.3.3, 4.5.18.3 and 4.5.20.3. In this case the broker will return data in the most concise lossless representation possible, for example removing all Attribute "type" members. When its value includes the keyword "keyValues" (or "simplified" as a synonym), a simplified representation of Entities shall be provided as defined by clause 4.5.4. If the Accept Header is set to "application/geo+json" the response will be in simplified GeoJSON format as defined by clause 4.5.17.

6.3.8 Notification behaviour

In the HTTP binding a notification that is triggered by a subscription shall be sent by issuing an HTTP POST request targeted to the value of "notification.endpoint.uri" member of the subscription structure (defined by clauses 5.2.12, 5.2.14 and 5.2.15). For the HTTP binding, the protocol part of the endpoint URI is http or https. In case the optional MQTT notification binding (clause 7) is supported, the protocol part of the endpoint URI can also be mqtt or mqtts. The MIME type associated to the POST request shall be "application/json" by default. However, this can be changed to "application/ld+json", or "application/geo+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], section 6.2 (to the default JSON-LD @context if none available).

If the optional array (of *KeyValuePair* type, as defined by clause 5.2.22) "notification.endpoint.receiverInfo" of the subscription is present, then a new custom HTTP header for each member named "key" of the key, value pairs that make up the array shall be generated and included in the HTTP POST's list of headers. The content of each custom header shall be set equal to the content of the corresponding "value" member of the *KeyValuePair*. "Key" and "value" members shall adhere to IETF RFC 7230 [27] Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing definitions concerning HTTP headers.

If the target MIME type is "application/geo+json" and the "notification.endpoint.receiverInfo" member contains a key "Prefer" whose value is set to "body=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], section 6.2 (to the default JSON-LD @context if none available).

If the target MIME type is "application/geo+json" and the "notification.endpoint.receiverInfo" contains a key "Prefer" whose value is set to "body=ld+json" or the "Prefer" key is omitted or "notification.endpoint.receiverInfo" does not exist, then the HTTP notification request includes an @context element in the payload body.

6.3.9 CSource Notification behaviour

In the HTTP binding a csource notification that is triggered by a csource subscription shall be sent by issuing an HTTP POST request targeted to the value of "notification.endpoint.uri" member of the csource subscription structure (defined by clauses 5.2.12 and 5.2.14). 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], section 6.2 (to the default JSON-LD @context if none available).

If the optional array (of *KeyValuePair* type, as defined by clause 5.2.22) "notification.endpoint.receiverInfo" of the subscription is present, then a new custom HTTP Header for each member named "key" of the key, value pairs that make up the array shall be generated and included in the HTTP POST's list of headers. The content of each custom header shall be set equal the content of the corresponding "value" member of the *KeyValuePair*. "Key" and "value" members shall adhere to IETF RFC 7230 [27] Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing definitions concerning HTTP headers.

6.3.10 Pagination behaviour

For HTTP operations corresponding to the operations listed in clause 4.12 (see table 6.2-1 for a list of HTTP operations with their corresponding clauses), implementations shall support the HTTP query parameter specified in table 6.3.10-1.

Table 6.3.10-1: Pagination: limit parameter

Name	Data Type	Cardinality	Remarks
limit	Integer	0..1	Only values greater or equal to 0. It defines the limit to the number of NGSI-LD Elements that shall be retrieved at a maximum as mandated by clause 5.5.9. The value 0 is only allowed in combination with the *count* URI parameter.

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

- The pointers to the next and previous pages (when needed as mandated by clause 5.5.9) shall be serialized as link-value elements. The content of such link-value(s) shall be:
  - For the next page, the Link Target shall be a URI-reference that could be dereferenced by an NGSI-LD Client to retrieve the next page of NGSI-LD Elements. In addition, the Link Relation Type shall be equal to "next", registered under the IANA Registry of Link Relation Types [20].

- For the previous page, the Link Target shall be a URI-reference that could be dereferenced by an NGSI-LD Client to retrieve the previous page of NGSI-LD Elements. In addition, the Link Relation Type shall be equal to "prev", registered under the IANA Registry of Link Relation Types [20].

- At least, the "type" Link Target Attribute shall be included by the previously described serialized Link Header, as mandated by IETF RFC 8288 [7], clause 3.4, and its value shall be exactly equal to the media type resulting from the original request made by the NGSI-LD Client (the request that triggered the current pagination iteration).

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

Temporal representation of resources adds an additional dimension to the pagination. Depending on the requested time range, the response will contain multiple instances of the requested Attribute, and therefore an additional pagination mechanism for those temporal representations is required, in order to limit the time range of the response. If no limits are specified, a default limit is enforced, depending on implementation specific configurations. For HTTP operations on temporal representations of Entities, implementations shall use the Partial Content Response (206) as specified by IETF RFC 7233 [31], clause 4.1, if the implementation is not able to respond with the full representation at once. In this case, for requests where the parameter "lastN" is present, pagination shall happen "backwards" (from the most recent to the least recent timestamp in the requested time range). For requests without the parameter "lastN", pagination shall happen "forwards" (from the least recent to the most recent timestamp in the requested time range).

This is achieved by including the "Content-Range" header field with the following contents:

- "unit" shall be equal to "DateTime";

- "range-start" and "range-end" shall be of type *DateTime*. They depend on the requested time relationship "timerel" (as defined by clause 4.11), as follows:

  - If the "lastN" parameter is present, pagination shall happen "backwards":

    - "range-start" shall be equal to "timeAt" for requests with "timerel=before", "endTimeAt" for requests with "timerel=between", or the most recent timestamp in the range of the response, for requests with "timerel=after";

    - "range-end" shall be equal to the least recent timestamp in the range of the response;

    - "size" shall be equal to the requested "lastN".

  - If the "lastN" parameter is **not** present, pagination shall happen "forwards":

    - "range-start" shall be equal to "timeAt" for requests with "timerel=after" or "timerel=between", or the least recent timestamp in the range of the response, for requests with "timerel=before";

    - "range-end" shall be equal to the most recent timestamp in the range of the response;

    - "size" shall be equal to "*".

6.3.11 Including system generated attributes

For HTTP GET operations performed over the resources /entities/, /subscriptions/, /csourceRegistrations/, /csourceSubscriptions/ and all of its sub-resources, implementations shall support the parameter specified in table 6.3.11-1. Implementations shall not raise an error if they do not hold system generated attributes.

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

Name	Data Type	Cardinality	Remarks
options	Comma separated list of strings	0..1	When its value includes the keyword "sysAttrs", a representation of NGSI-LD Elements shall be provided so that the system generated attributes *createdAt*, *modifiedAt* are included in the response payload body where known. In the case of temporal representations, also the system generated attribute *deletedAt* is included, if the NGSI-LD Element has been deleted.

6.3.12 Simplified or aggregated temporal representation of entities

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

Table 6.3.12-1: Simplified representation: options parameter

Name	Data Type	Cardinality	Remarks
options	Comma separated list of strings	0..1	When its value includes the keyword "temporalValues", a simplified temporal representation of entities shall be provided as defined by clause 4.5.8. When its value includes the keyword "aggregatedValues", an aggregated temporal representation of entities shall be provided as defined by clause 4.5.19. Only one of the two keywords can be present in the values of the parameter.

6.3.13 Counting number of results

This clause implements the behaviour described in clause 4.13, in case of HTTP binding.

For HTTP operations corresponding to the operations listed in clause 4.12 (see table 6.2-1 for a list of HTTP operations with their corresponding clauses), implementations shall support the HTTP query parameter specified in table 6.3.13-1.

Table 6.3.13-1: Counting number of results: count parameter

Name	Data Type	Cardinality	Remarks
count	Boolean	0..1	If *true*, then a special HTTP header (NGSILD-Results-Count) is set in the response. Regardless of how many entities are actually returned (maybe due to the "limit" URI parameter), the total number of matching results (e.g. number of Entities) is returned.

This clause defines the specific HTTP binding mechanisms that can be useful to plan the "limit" and "offset" URI parameters for pagination, thus allowing to convey an overview of the number of entities in a system.

To get only the count itself, and no entities, the URI parameter "limit" may have the value "0", and an empty array shall be returned as payload body.

Setting the URI parameter "limit" to zero without including the "count" URI parameter will result in a 400 Bad Request error.

6.3.14 Tenant specification

If the system implementing the NGSI-LD API supports multi-tenancy as described in clause 4.14 and clause 5.5.10, the tenant, to which the NGSI-LD HTTP operation is targeted, is specified as the HTTP header "NGSILD-Tenant", whose value is the String identifying the tenant. In case the target tenant is the default tenant, the HTTP header is omitted. If the HTTP header "NGSILD-Tenant" is present in the HTTP request, it shall also be present in HTTP response. This also applies to HTTP notifications sent as a result of subscriptions with an "NGSILD-Tenant" HTTP header (clause 6.3.8).

6.3.15 GeoJSON representation of spatially bound entities

For HTTP GET operations performed over the resource /entities and /entities/{entity-id}, if the GeoJSON Accept header ("application/geo+json") is present, implementations shall render the entities of the response in the GeoJSON format, as described in clause 5.2.29.

For GeoJSON representations, a GeoProperty may be selected as the geolocation to be used as the geometry within the GeoJSON payload. If no "geometryProperty" parameter is specified then the "location" GeoProperty of the Entity is used.

Table 6.3.15-1: Selecting a geometry

Name	Data Type	Cardinality	Remarks
geometryProperty	String	0..1	If not present, "location" is used.
datasetId	String	0..1	Shall be valid URI. If the referenced GeoProperty consists of an attribute with multiple instances the datasetId specifies which instance of the value is to be selected. If not present, the default instance is used.

6.3.16 Expiration time for cached @contexts

Implementations shall comply with the Expires header field (section 5.3 of IETF RFC 7234 [30]) or a max-age or s-maxage response directive of Cache-Control header field (section 5.2.2 of IETF RFC 7234 [30]) that may be present in the downloaded @context. This means that implementations shall periodically invalidate the "Cached" @contexts according to the headers mentioned above. Since origin servers do not always provide explicit expiration times, implementations should assign a heuristic (for instance according to IETF RFC 7234 [30] section 4.2.2) expiration time when an explicit time is not specified.

6.3.17 Distributed Operations Caching and Timeout Behaviour

The caching of response data received from forwarded HTTP GET requests is optionally supported by Context Brokers. For HTTP GET operations performed over the resources /entities and /entities/{entity-id}, where a Context Source Registration matches the request and a previous forwarded response has been cached and a subsequent request occurs before the Context Source Registration "cacheDuration" (as defined in table 5.2.34-1) has been reached, the result may incorporate data cached from a previous response. To indicate that data from a cache has been included in the response, the HTTP header "NGSILD-Warning" shall be included. The value shall match the IANA Warning Code [32]
110 - Response is Stale.

"NGSILD-Warning" HTTP headers shall also be used to indicate instances of abnormal behaviour for distributed HTTP GET operations performed over the resources /entities and /entities/{entity-id}.

Table 6.3.17-1: NGSILD Warning Codes

IANA Warning Code	Remarks
110 - Response is Stale	No request was made to a specified registration endpoint - data from a cached value has been reused and it has been incorporated into the response.
111 - Revalidation Failed	Although data was received from the registration endpoint within the specified timeout period, the payload of the response was invalid. This could occur if the payload was corrupted or a non-NGSI payload was received.
199 - Miscellaneous Warning	No response was received from the registration endpoint within the specified timeout period.
299 - Miscellaneous Persistent Warning	An error response (such as **403 - Forbidden**) was received from the registration endpoint within the specified timeout period. This could occur if the Context Broker has insufficient access rights to retrieve the data.

For distributed HTTP GET operations, registered context sources should always respond with a valid NGSI-LD payload. The Context Broker shall successfully parse this data and invalid non-NGSI-LD payloads shall be rejected and not incorporated into the overall response. It should be noted that a registration endpoint responding with no data and the HTTP status code **404 - Not Found** should not be considered as abnormal behaviour for distributed operations.

For all other operations, which correspond to HTTP Unsafe methods, the error response should be as informative as possible.

In the case of an **exclusive** or **redirect** registration, where all of the data is held outside of the Context Broker and held in a single registered source:

- 504 Gateway Timeout - if the single registered source fails to respond in time.
- 404 Not Found - if resources not found within the single registered source.

- 502 Bad Gateway - if the single forwarded request fails for any other reason such as the Context Broker itself having insufficient access rights.

In the case of an **inclusive** or **redirect** registration, where an entity is distributed over multiple equally valid endpoints, but when updating the state of the distributed entity, an error response is returned from one or more registered sources:

- 207 Multi Status

In the case of an **auxiliary** registration HTTP unsafe methods are not supported.

Whenever failures or timeouts occur, Context Brokers may optionally decline to make subsequent requests to the same registration endpoint until the cooldown period (as defined in table 5.2.9-1) has been reached.

6.3.18 Limiting Distributed Operations

This clause amends the matching Context Source Registrations behaviour as described in clause 5.12, in the case of the HTTP binding in order to avoid cascading distributed operations (see clause 4.3.6.4). For all operations the resources /entities/, /types/, /attributes/, /subscriptions/, /csourceSubscriptions/, /entityOperations/, /temporal/entities/ and temporal/entityOperations/ implementations shall support the HTTP query parameter specified in table 6.3.18-1.

Table 6.3.18-1: Limiting distributed operations: local parameter

Name	Data Type	Cardinality	Remarks
local	Boolean	0..1	If local=true then no Context Source Registrations shall be considered as matching to avoid cascading distributed operations (see clause 4.3.6.4)

6.3.19 Extra information to provide when contacting Context Source

As specified in clauses 4.3.6.5 and 4.3.6.6, extra information to be provided when contacting a Context Source can be specified in the optional array (of KeyValuePair type, as defined by clause 5.2.22) "contextSourceInfo" of the CSourceRegistration.

In the HTTP binding, if the "jsonldContext" key is present, the URL value is placed in an HTTP "Link" Header as described by the JSON-LD specification [2], section 6.2 and, whenever a payload body is present in the request, the HTTP "Content-Type" Header is set to "application/json". For all other keys, a new custom HTTP header is added for each member named "key" of the key-value pairs that make up the array shall be generated and included in the HTTP list of headers. The content of each custom header shall be set equal to the content of the corresponding "value" member of the KeyValuePair, unless the special value "urn:ngsi-ld:request" has been set, in which case the value is to be taken from the triggering request, if present there. "Key" and "value" members shall adhere to IETF RFC 7230 [27] Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing definitions concerning HTTP headers.

Headers derived from other elements of the CSourceRegistration, e.g. "NGSILD-Tenant", take precedence and cannot be overridden using contextSourceInfo. The same applies for headers generally set by HTTP itself, e.g. Content-length.

6.3.20 Invalid parameters

If an HTTP request for an operation contains parameters that are incompatible with the operation, or it contains values of the "options" parameter that are not supported by the operation, an HTTP error response of type *InvalidRequest* should be returned.

6.4 Resource: entities/

6.4.1 Description

This resource represents the entities known to an NGSI-LD system.

6.4.2 Resource definition

Resource URI:

- /entities/

6.4.3 Resource methods

6.4.3.1 POST

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

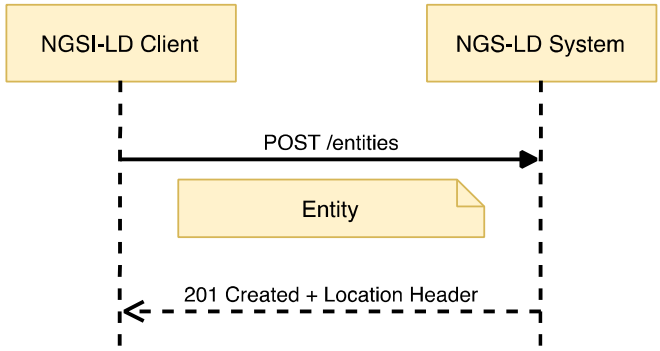

Figure 6.4.3.1-1: Create Entity interaction

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

	Data Type	Cardinality	Remarks	
Request Body	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	The HTTP response shall include a "Location" HTTP header that contains the resource URI of the created entity resource.
	BatchOperationResult	1	207 Multi-Status	The HTTP response shall include a "Location" HTTP header that contains the resource URI of the created entity resource. If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	409 Conflict	It is used to indicate that the entity or an exclusive or redirect registration defining 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 (see IETF RFC 7807 [10])	1	422 Unprocessable Entity	It is used to indicate that the operation is not available, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.4.3.2 GET

This method is associated to the operation "Query Entities" and shall exhibit the behaviour defined by clause 5.7.2, providing entities as part of the HTTP response payload body. In addition to this method, an alternative way to perform "Query Entities" operations via POST is defined in clause 6.23. Figure 6.4.3.2-1 shows the query entities interaction.

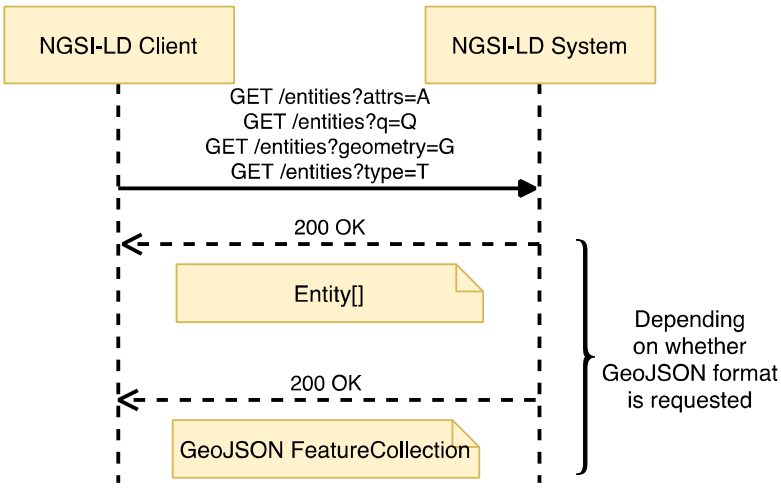

Figure 6.4.3.2-1: Query Entities interaction

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 strings	0..1	Each String shall be a valid URI. List of entity ids to be retrieved.
type	String	0..1 At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Selection of Entity Types as per clause 4.17.
idPattern	Regular expression as defined by [11]	0..1	Regular expression that shall be matched by entity ids.
attrs	Comma separated list of strings	0..1 At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Each String is an Attribute (Property or Relationship) name. List of Attributes to be matched by the Entities and also included in the response, i.e. only Entities that contain at least one of the Attributes in *attrs* are to be included in the response, and only the Attributes listed in *attrs* are to be included in each of the Entities of the response.
q	String	0..1 At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Query as per clause 4.9.
expandValues	Comma separated list of strings	0..1	Each String is an Attribute (Property or Relationship) name. List of Attributes whose values shall be expanded into URIs according to the supplied @context prior to executing a query. It is part of query.
csf	String	0..1	Context Source filter as per clause 4.9.
geometry	String	0..1 It shall be 1 if *georel* or *coordinates* are present. At least one among: *type*, *attrs*, *q*, or *geometry* shall be present.	Geometry as per clause 4.10. It is part of geoquery.
georel	String	0..1 It shall be 1 if *geometry* or *coordinates* are present.	Geo relationship as per clause 4.10. It is part of geoquery.
coordinates	String	0..1 It shall be one if *geometry* or *georel* are present.	Coordinates serialized as a string as per clause 4.10. It is part of geoquery.

Name	Data Type	Cardinality	Remarks
geoproperty	String	0..1 It shall be ignored unless a geoquery is present.	It represents the name of the Property that contains the geospatial data that will be used to resolve the geoquery. By default, will be *location* (see clause 4.7).
geometryProperty	String	0..1	It represents a Property name. In the case of GeoJSON Entity representation, this parameter indicates which GeoProperty to use for the toplevel *geometry* field.
lang	String	0..1	It represents the preferred natural language of the response. It is used to reduce languageMaps to a string or string array property in a single preferred language.
scopeQ	String	0..1	Scope query (see clause 4.19).

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

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Entity[]	1	200 OK	A response body containing the query result as a list of entities, unless the Accept Header indicates that the Entities are to be rendered as GeoJSON.
	GeoJSON FeatureCollection	1	200 OK	If the Accept Header indicates that the Entities are to be rendered as GeoJSON, a response body containing the query result as GeoJSON FeatureCollection is returned.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	501 Not Implemented	It is used by Registered Context Sources to indicate that the data format of the request is unsupported see clause 6.3.7.

6.5 Resource: entities/{entityId}

6.5.1 Description

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

6.5.2 Resource definition

Resource URI:

- /entities/{entityId}

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

Table 6.5.2-1: URI variables

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

6.5.3 Resource methods

6.5.3.1 GET

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

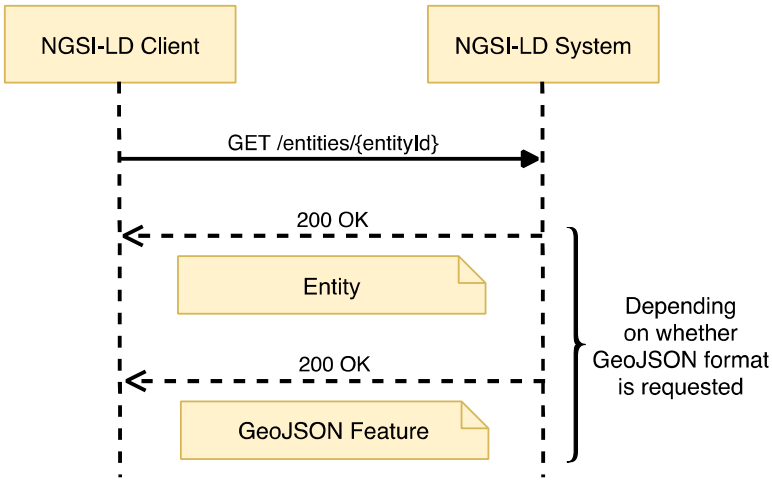

Figure 6.5.3.1-1: Retrieve Entity interaction

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

Table 6.5.3.1-1: Query parameters

Name	Data Type	Cardinality	Remarks
attrs	Comma separated list of strings	0..1	Each String is an Attribute (Property or Relationship) name. List of Attributes to be matched by the Entity and included in the response. If the Entity does not have any of the Attributes in *attrs*, then a *404 Not Found* shall be retrieved. If *attrs* is not specified, no matching is performed and all Attributes related to the Entity shall be retrieved.
type	String	0..1	Selection of Entity Types as per clause 4.17.
geometryProperty	String	0..1	It represents a GeoProperty name. In the case of GeoJSON Entity representation, this parameter indicates which GeoProperty to use for the "geometry" element. By default, it shall be 'location'.
lang	String	0..1	It represents the preferred natural language of the response. It is used to reduce languageMaps to a string or string array property in a single preferred language.

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

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Entity	1	200 OK	A response body containing the JSON-LD representation of the target entity containing the selected Attributes, unless the Accept Header indicates that the Entity is to be rendered as GeoJSON.
	GeoJSON Feature	1	200 OK	If the Accept Header indicates that the Entity is to be rendered as GeoJSON, a GeoJSON Feature is returned.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.
	ProblemDetails (see IETF RFC 7807 [10])	1	501 Not Implemented	It is used by Registered Context Sources to indicate that the data format of the request is unsupported see clause 6.3.7.

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.

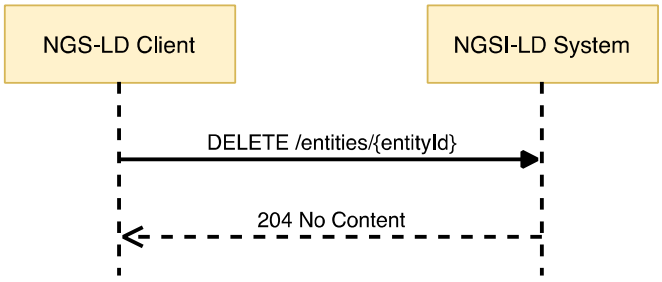

Figure 6.5.3.2-1: Delete Entity interaction

The query parameters that shall be supported are those defined in table 6.5.3.2-1 and table 6.5.3.2-2 describes the request body and possible responses.

Table 6.5.3.2-1: Query parameters

Name	Data Type	Cardinality	Remarks
type	String	0..1	Selection of Entity Types as per clause 4.17

Table 6.5.3.2-2: Delete Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	BatchOperation Result	1	207 Multi-Status	If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

6.5.3.3 PUT

This method is bound to the "Replace Entity" operation and shall exhibit the behaviour defined by clause 5.6.18. The entity identifier is the value of the resource URI variable "entityId". The data to be updated shall be contained in the HTTP request payload body. Figure 6.5.3.3-1 shows the Replace Entity interaction.

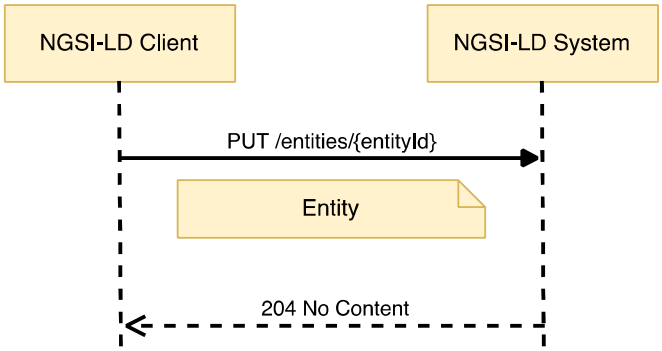

Figure 6.5.3.3-1: Replace Entity interaction

The query parameters that shall be supported are those defined in table 6.5.3.3-1 and table 6.5.3.3-2 describes the request body and possible responses.

Table 6.5.3.3-1: Query parameters

Name	Data Type	Cardinality	Remarks
type	String	0..1	Selection of Entity Types as per clause 4.17

Table 6.5.3.3-2: Replace Entity request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity Fragment	1	Entity Fragment containing a complete representation of the Entity to be replaced.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No content	The entity was replaced successfully.
	BatchOperationResult	1	207 Multi-Status	If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier not known to the system, see clause 6.3.2.

6.5.3.4 PATCH

This method is bound to the "Merge Entity" operation and shall exhibit the behaviour defined by clause 5.6.17. The entity identifier is the value of the resource URI variable "entityId". The data to be updated shall be contained in the HTTP request payload body. Figure 6.5.3.4-1 shows the Merge Entity interaction.

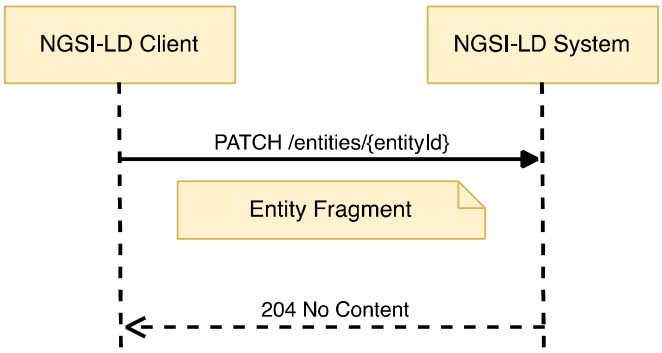

Figure 6.5.3.4-1: Merge Entity interaction

The query parameters that shall be supported are those defined in table 6.5.3.4-1 and table 6.5.3.4-2 describes the request body and possible responses.

Table 6.5.3.4-1: Query parameters

Name	Data Type	Cardinality	Remarks
options	Comma separated list of strings	0..1	When its value includes the keyword "keyValues" (or "simplified" as a synonym), this indicates that a simplified representation of Entities has been provided as defined by clause 4.5.4. In this case, when a merge operation applies to an existing Attribute the "type" attribute shall remain unchanged.
type	String	0..1	Selection of Entity Types as per clause 4.17.
observedAt	String	0..1	It shall be a DateTime (see clause 4.6.3). When a merge operation applies to a pre-existing Attribute which previously contained an "observedAt" sub-attribute, the value held in this query parameter shall be used if no specific "observedAt" sub-Attribute is found in the payload body.
lang	String	0..1	It represents the natural language of data held in the request. When a merge operation applies to a pre-existing LanguageProperty and the value is supplied as a string or string array in the payload body, this query parameter shall be used to determine the key within the languageMap JSON Object to update.

Table 6.5.3.4-2: Merge Entity request body and possible responses

	Data Type	Cardinality	Remarks	
Request Body	Entity Fragment	1	Entity Fragment containing a complete representation of the Attributes to be merged.	
	Data Type	**Cardinality**	**Response Codes**	**Remarks**
Response Body	N/A	N/A	204 No content	All the Attributes were merged successfully.
	BatchOperationResult	1	207 Multi-Status	If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier not known to the system, see clause 6.3.2.

6.6 Resource: entities/{entityId}/attrs/

6.6.1 Description

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

6.6.2 Resource definition

Resource URI:

- /entities/{entityId}/attrs

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

Table 6.6.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity

6.6.3 Resource methods

6.6.3.1 POST

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

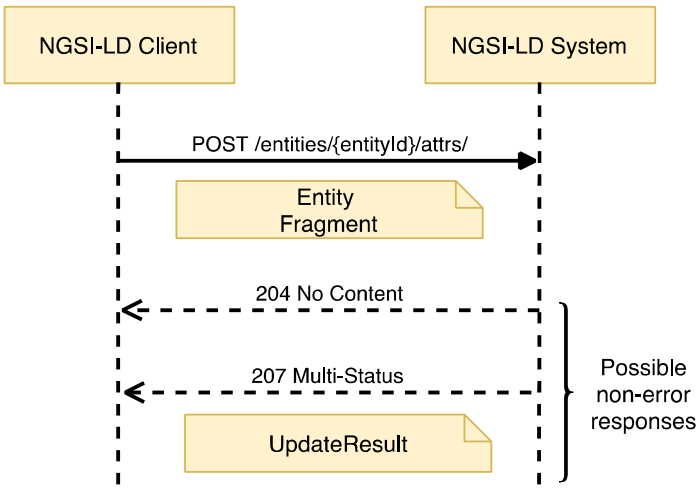

Figure 6.6.3.1-1: Append Attributes interaction

The query parameters that shall be supported are those defined in table 6.6.3.1-1 and table 6.6.3.1-2 describes the request body and possible responses.

Table 6.6.3.1-1: Query parameters

Name	Data Type	Cardinality	Remarks
type	String	0..1	Selection of Entity Types as per clause 4.17.
options	Comma separated list of strings	0..1	"noOverwrite" indicates that no attribute overwrite shall be performed.

Table 6.6.3.1-2: Post Attributes request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	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	N/A	204 No content	All the Attributes were appended successfully.
	UpdateResult	1	207 Multi-Status	Only the Attributes included in the response payload body were successfully appended. If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a UpdateResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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 Attributes" operation and shall exhibit the behaviour defined by clause 5.6.2. The entity identifier is the value of the resource URI variable "entityId". The data to be updated shall be contained in the HTTP request payload body. Figure 6.6.3.2-1 shows the Update Attributes interaction.

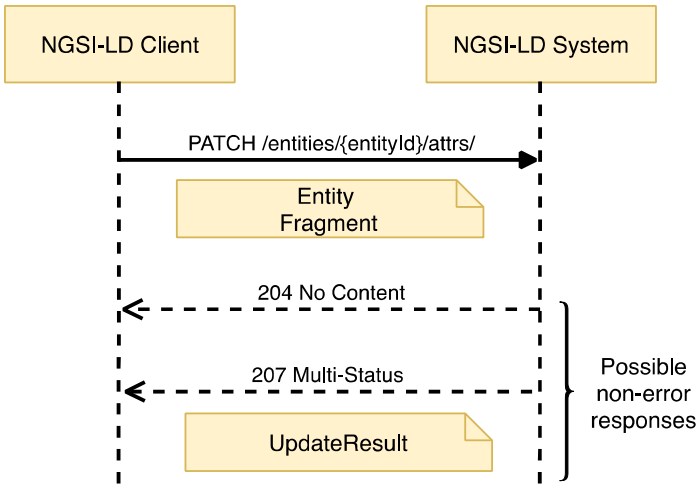

Figure 6.6.3.2-1: Update Attributes interaction

The query parameters that shall be supported are those defined in table 6.6.3.2-1 and table 6.6.3.2-2 describes the request body and possible responses.

Table 6.6.3.2-1: Query parameters

Name	Data Type	Cardinality	Remarks
type	String	0..1	Selection of Entity Types as per clause 4.17

Table 6.6.3.2-2: Patch Attributes request body and possible responses

	Data Type	Cardinality	Remarks	
Request Body	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	N/A	204 No content	All the Attributes were updated successfully.
	UpdateResult	1	207 Multi-Status	Only the Attributes included in the response payload body were successfully updated. If no Attributes were successfully updated the *updated* array of *UpdateResult* (see clause 5.2.18) will be empty. If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier not known to the system, see clause 6.3.2.

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

6.7.1 Description

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

6.7.2 Resource definition

Resource URI:

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

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

Table 6.7.2-1: URI variables

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

6.7.3 Resource methods

6.7.3.1 PATCH

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

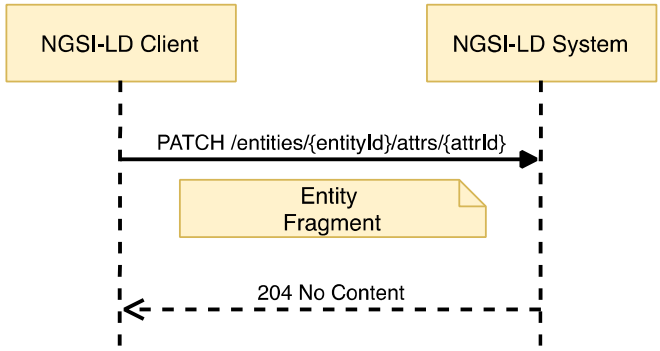

Figure 6.7.3.1-1: Partial Attribute Update interaction

The query parameters that shall be supported are those defined in table 6.7.3.1-1 and table 6.7.3.2-2 describes the request body and possible responses.

Table 6.7.3.1-1: Query parameters

Name	Data Type	Cardinality	Remarks
type	String	0..1	Selection of Entity Types as per clause 4.17

Table 6.7.3.1-2: Partial Attribute Update request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity Fragment	1	Entity Fragment containing the elements of the attribute to be updated.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	The attribute was updated successfully.
	UpdateResult	1	207 Multi-Status	If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a UpdateResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.

	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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 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 Attribute interaction, table 6.7.3.2-1 shows the delete parameters to be supported and table 6.7.3.2-2 describes the request body and possible responses.

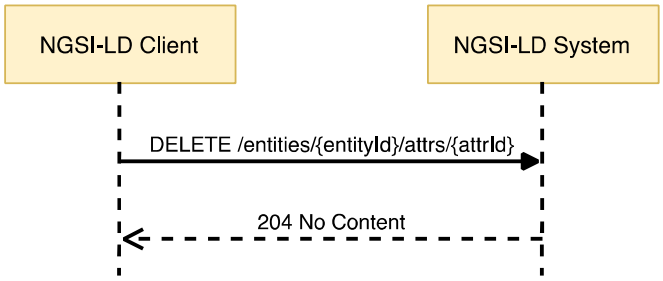

Figure 6.7.3.2-1: Delete Attribute interaction

Table 6.7.3.2-1: Delete parameters

Name	Data Type	Cardinality	Remarks
deleteAll	Boolean	0..1	If *true*, all attribute instances are deleted. Otherwise (default) only the Attribute instance specified by the *datasetId* is deleted. In case neither the deleteAll flag nor a *datasetId* is present, the default Attribute instance is deleted.
type	String	0..1	Selection of Entity Types as per clause 4.17.
datasetId	String	0..1	Shall be a valid URI. Specifies the *datasetId* of the dataset to be deleted.

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	UpdateResult	1	207 Multi-Status	If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a UpdateResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.

	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

6.7.3.3 PUT

This method is bound to the "Attribute replace" operation and shall exhibit the behaviour defined by clause 5.6.19. 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 Attribute Fragment shall be contained in the HTTP request payload body. Figure 6.7.3.3-1 shows the Attribute replace interaction.

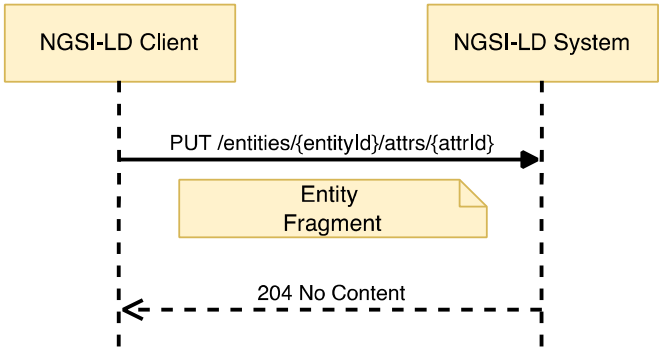

Figure 6.7.3.3-1: Attribute replace interaction

The query parameters that shall be supported are those defined in table 6.7.3.3-1 and table 6.7.3.3-2 describes the request body and possible responses.

Table 6.7.3.3-1: Query parameters

Name	Data Type	Cardinality	Remarks
type	String	0..1	Selection of Entity Types as per clause 4.17

Table 6.7.3.3-2: Attribute replace request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	Attribute Fragment	1	Attribute Fragment replacing the previous data.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	The attribute was replaced successfully.
	UpdateResult	1	207 Multi-Status	If the entity input data matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a UpdateResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

6.8 Resource: csourceRegistrations/

6.8.1 Description

This resource represents the context source registrations known to an NGSI-LD system.

6.8.2 Resource definition

Resource URI:

- /csourceRegistrations/

6.8.3 Resource methods

6.8.3.1 POST

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

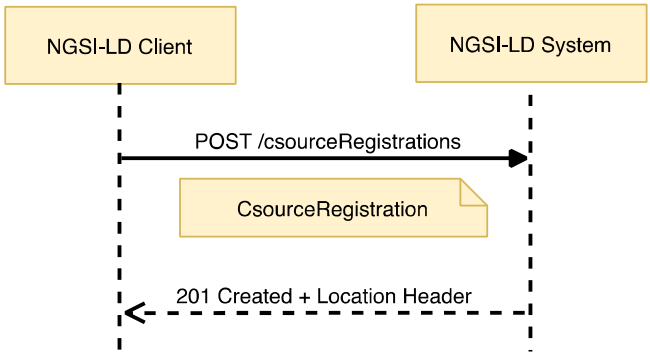

Figure 6.8.3.1-1: Register Context Source interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	CSourceRegistration	1	Payload body in the request contains a JSON-LD object which represents the context source registration that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	The HTTP response shall include a "Location" HTTP header that contains the resource URI of the created context source registration resource.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	409 Conflict	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 (see IETF RFC 7807 [10])	1	422 Unprocessable Context Source Registration	It is used to indicate that the operation is not available see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.8.3.2 GET

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

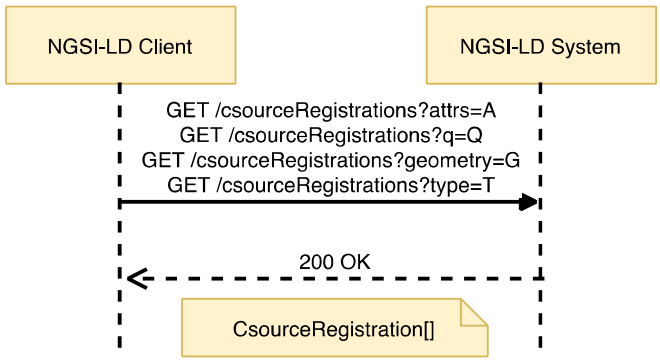

Figure 6.8.3.2-1: Query Context Source Registrations interaction

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

Table 6.8.3.2-1: Query parameters

Name	Data Type	Cardinality	Remarks
id	Comma separated list of strings	0..1	Each String shall be a valid URI. List of entity ids to be retrieved
type	String	0..1 At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Selection of Entity Types as per clause 4.17
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 strings	0..1 At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Each String is an Attribute (Property or Relationship) name. List of Attributes (Properties or Relationships) to be retrieved
q	String	0..1 At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Query as per clause 4.9
csf	String	0..1	Context Source filter as per clause 4.9
geometry	String	0..1 It shall be 1 if *georel* or *coordinates* are present. At least one among: *type*, *attrs*, *q*, or *georel* shall be present.	Geometry as per clause 4.10. It is part of geoquery
georel	String	0..1 It shall be 1 if *geometry* or *coordinates* are present.	Geo relationship as per clause 4.10. It is part of geoquery
coordinates	String	0..1 It shall be one if *geometry* or *georel* are present.	Coordinates serialized as a string as per clause 4.10. It is part of geoquery
geoproperty	String	0..1 It shall be ignored if no geoquery is present.	It represents the name of the Property that contains the geospatial data that will be used to resolve the geoquery
timeproperty	String	0..1 It shall be ignored if no temporal query is present.	It represents a Temporal Property name Allowed values: "observedAt", "createdAt", "modifiedAt" and "deletedAt". If not specified, the default is "observedAt". (See clause 4.8)
timerel	String	0..1	It represents the temporal relationship as defined by clause 4.1 Allowed values: "before", "after", "between"

Name	Data Type	Cardinality	Remarks
timeAt	String	0..1	It represents the *timeAt* parameter as defined by clause 4.1 It shall be a *DateTime*. Cardinality shall be 1 if *timerel* is present
endTimeAt	String	0..1	It represents the *endTimeAt* parameter as defined by clause 4.1 It shall be a *DateTime.* Cardinality shall be 1 if *timerel* is equal to "between"
geometryProperty	String	0..1	It represents a Property name In the case of GeoJSON Entity representation, this parameter indicates which GeoProperty to use for the toplevel *geometry* field
lang	String	0..1	It represents the preferred natural language of the response It is used to reduce languageMaps to a string or string array property in a single preferred language
scopeQ	String	0..1	Scope query (see clause 4.19)

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	CSourceRegistration[]	1	200 OK	A response body containing the query result as an array of context source registrations.
	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.9 Resource: csourceRegistrations/{registrationId}

6.9.1 Description

This resource represents the context source registration, identified by *registrationId*, known to an NGSI-LD system.

6.9.2 Resource definition

Resource URI:

- /csourceRegistrations/{registrationId}

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

Table 6.9.2-1: URI variables

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

6.9.3 Resource methods

6.9.3.1 GET

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

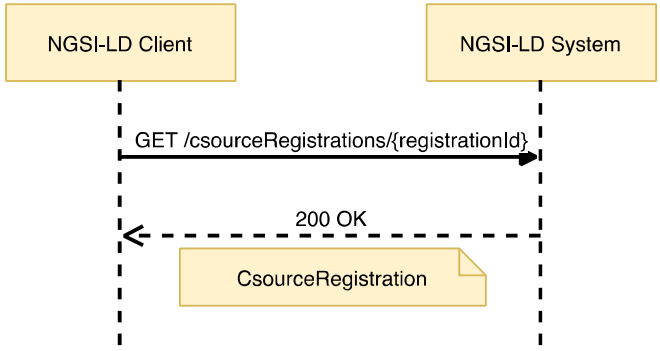

Figure 6.9.3.1-1: Retrieve Context Source Registration interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	CSourceRegistration	1	200 OK	A response body containing the JSON-LD representation of the target context source registration.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

6.9.3.2 PATCH

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

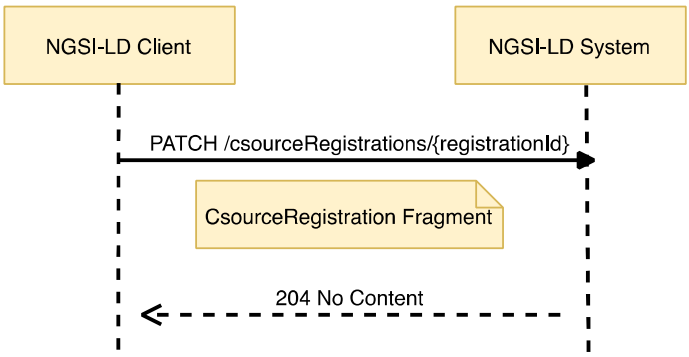

Figure 6.9.3.2-1: Update Context Source Registration interaction

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

	Data Type	Cardinality	Remarks	
Request Body	CSourceRegistration Fragment	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	N/A	204 No Content	The context source registration was successfully updated.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

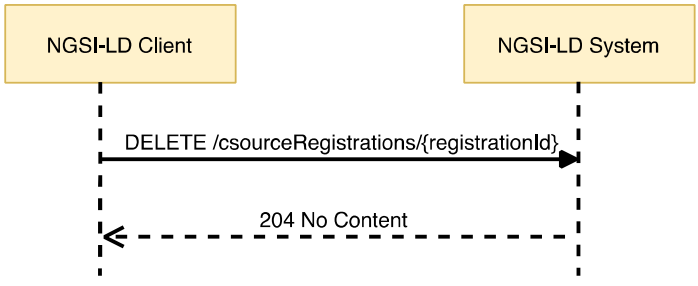

Figure 6.9.3.3-1: Delete Context Source Registration interaction

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

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	Data Type	Cardinality	Response Codes	Remarks
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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 the subscriptions known to an NGSI-LD system.

6.10.2 Resource definition

Resource URI:

- /subscriptions/

6.10.3 Resource methods

6.10.3.1 POST

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

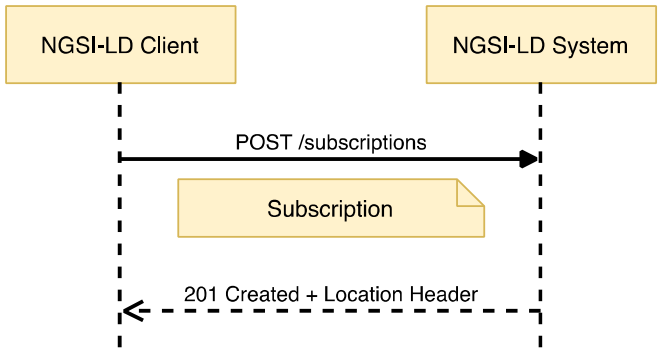

Figure 6.10.3.1-1: Create Subscription interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Subscription	1	Payload body in the request contains a JSON-LD object which represents the subscription that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	The HTTP response shall include a "Location" HTTP header that contains the resource URI of the created subscription resource.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	409 Conflict	It is used to indicate that the subscription already exists see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.10.3.2 GET

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

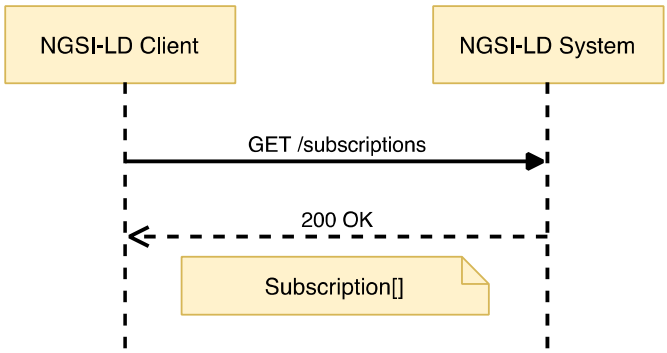

Figure 6.10.3.2-1: Query Subscriptions interaction

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

Table 6.10.3.2-1: Query parameters

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

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription[]	1	200 OK	A response body containing a list of subscriptions.
	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.11 Resource: subscriptions/{subscriptionId}

6.11.1 Description

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

6.11.2 Resource definition

Resource URI:

- /subscriptions/{subscriptionId}

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

Table 6.11.2-1: URI variables

Name	Definition
subscriptionId	Id (URI) of the concerned subscription

6.11.3 Resource methods

6.11.3.1 GET

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

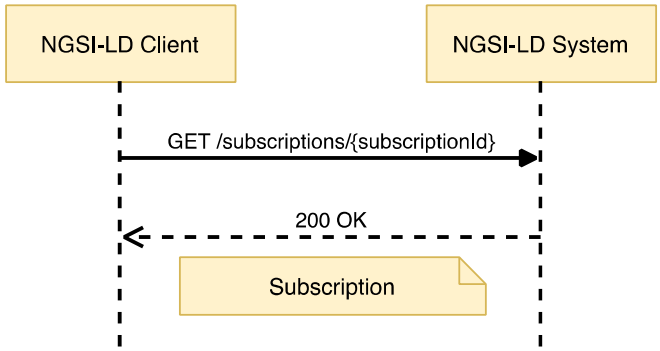

Figure 6.11.3.1-1: Retrieve Subscription interaction

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

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription	1	200 OK	A response body containing the JSON-LD representation of the target subscription.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

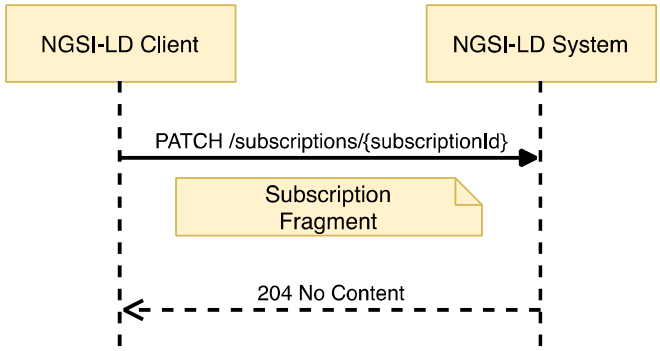

Figure 6.11.3.2-1: Update Subscription interaction

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	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

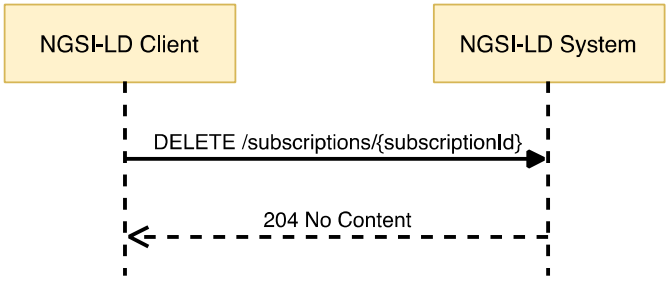

Figure 6.11.3.3-1: Delete Subscription interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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 the context source registration subscriptions known to an NGSI-LD system.

6.12.2 Resource definition

Resource URI:

- /csourceSubscriptions/

6.12.3 Resource methods

6.12.3.1 POST

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

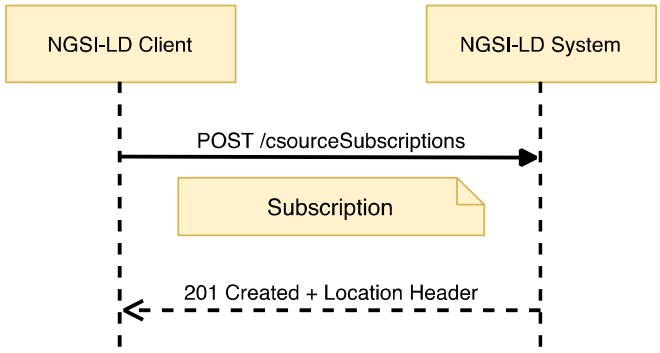

Figure 6.12.3.1-1: Create Context Source Registration Subscription interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Subscription	1	Payload body in the request contains a JSON-LD object which represents the context source registration subscription that is to be created.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	The HTTP response shall include a "Location" HTTP header that contains the resource URI of the created context source registration subscription resource.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	409 Conflict	It is used to indicate that the context source registration subscription already exists, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.12.3.2 GET

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

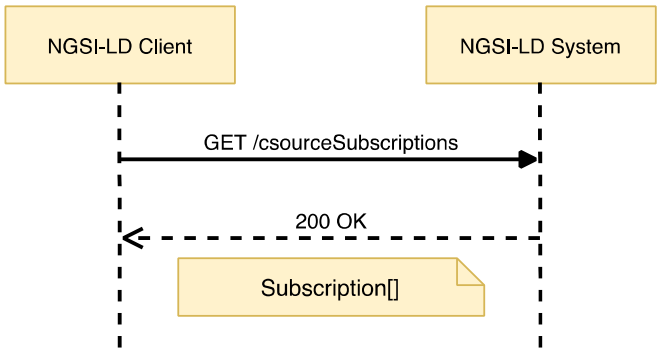

Figure 6.12.3.2-1: Query Context Source Registration Subscriptions interaction

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

Table 6.12.3.2-1: Query parameters

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

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription[]	1	200 OK	A response body containing a list of context source registration subscriptions.
	ProblemDetails (see IETF RFC 7807 [10])	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 the context source registration subscription, identified by *subscriptionId*, known to an NGSI-LD system.

6.13.2 Resource definition

Resource URI:

- /csourceSubscriptions/{subscriptionId}

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

Table 6.13.2-1: URI variables

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

6.13.3 Resource methods

6.13.3.1 GET

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

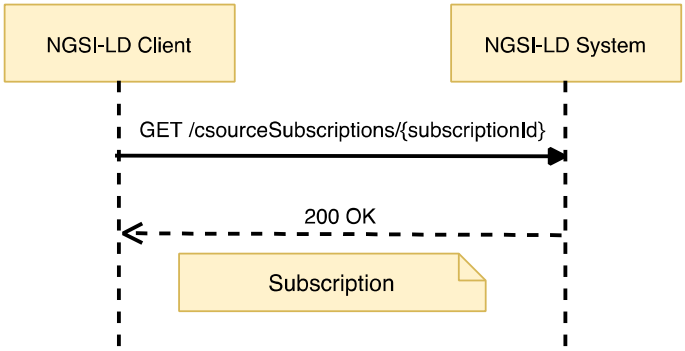

Figure 6.13.3.1-1: Retrieve Context Source Registration Subscription interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Subscription	1	200 OK	A response body containing the JSON-LD representation of the target context source registration subscription.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

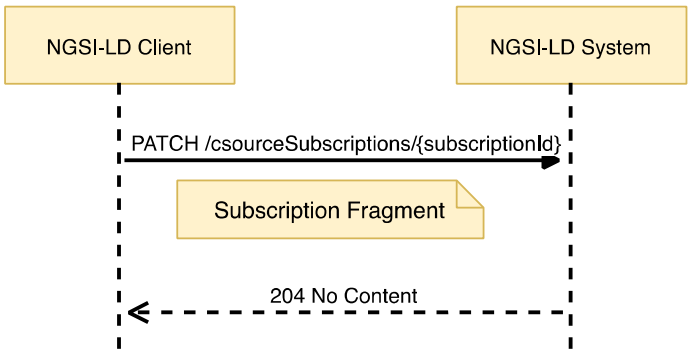

Figure 6.13.3.2-1: Update Context Source Registration Subscription interaction

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	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	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.

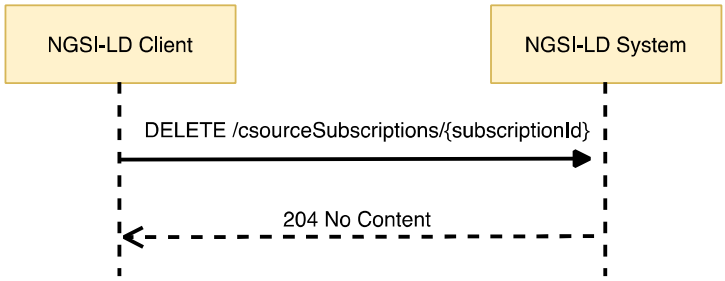

Figure 6.13.3.3-1: Delete Context Source Registration Subscription interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided a subscription identifier (URI) not known to the system, see clause 6.3.2.

6.14 Resource: entityOperations/create

6.14.1 Description

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

6.14.2 Resource definition

Resource URI:

- /entityOperations/create

6.14.3 Resource methods

6.14.3.1 POST

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

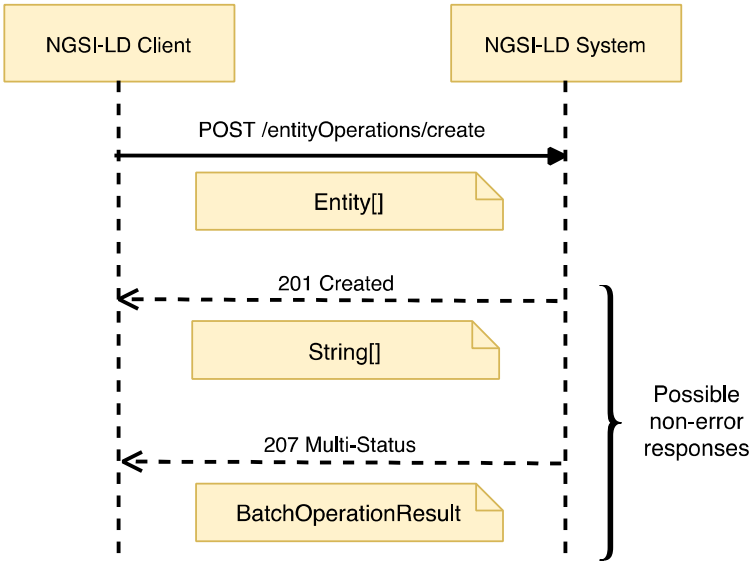

Figure 6.14.3.1-1: Batch Entity Creation Interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of entities to be created	
Response Body	**Data Type**	**Cardinality**	**Response Code**	**Remarks**
	String[]	1	201 Created	If all entities have been successfully created, an array of Strings containing URIs is returned in the response. Each URI represents the Entity Id of a created entity. There is no restriction as to the order of the Entity Ids.
	BatchOperationResult	1	207 Multi-Status	If only some or none of the entities have been successfully created, a response body containing the result of each operation contained in the batch is returned in a BatchOperationResult structure. It contains two arrays. The first array ('success') contains the URIs of the successfully created entities, while the second array ('errors') contains information about the error for each of the entities that could not be created. There is no restriction as to the order of the Entity Ids in the arrays. If any of the entities matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.15 Resource: entityOperations/upsert

6.15.1 Description

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

6.15.2 Resource definition

Resource URI:

- /entityOperations/upsert

6.15.3 Resource methods

6.15.3.1 POST

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

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

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

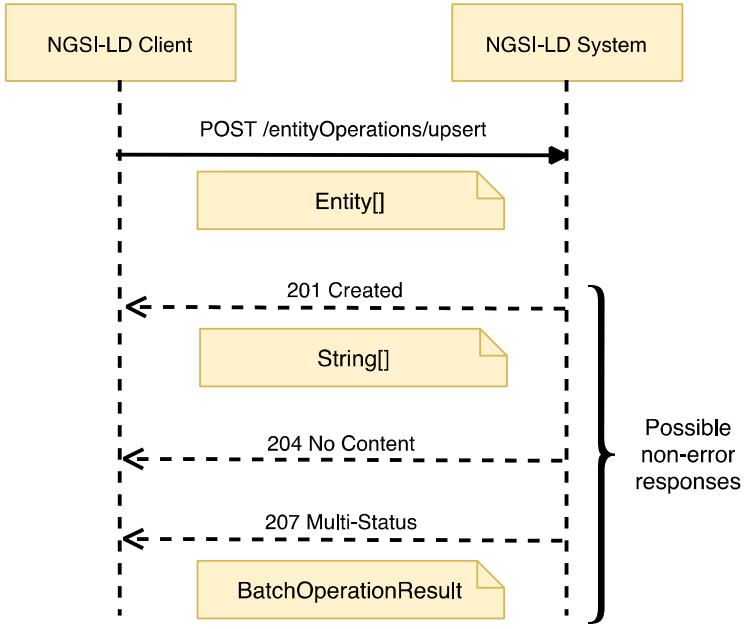

Figure 6.15.3.1-1: Batch Entity Creation or Update Interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of entities to be created/updated	
Response Body	**Data Type**	**Cardinality**	**Response Code**	**Remarks**
	String[]	1	201 Created	If all entities not existing prior to this request have been successfully created and the others have been successfully updated, an array of String (with the URIs representing the Entity Ids of the created entities only) is returned in the response. There is no restriction as to the order of the Entity Ids. The merely updated entities do not take part in the response (corresponding to 204 No Content returned in the case of updates).
	N/A	N/A	204 No Content	If all entities already existed and are successfully updated, there is no payload body in the response.
	BatchOperationResult	1	207 Multi-Status	If only some or none of the entities have been successfully created or updated, a response body containing the result of each operation contained in the batch is returned in a BatchOperationResult structure. It contains two arrays. The first array ('success') contains the URIs of the successfully created or updated entities, while the second array ('errors') contains information about the error for each of the entities that could not be created or updated. There is no restriction as to the order of the Entity Ids in the arrays. If any of the entities matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.16 Resource: entityOperations/update

6.16.1 Description

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

6.16.2 Resource definition

Resource URI:

- /entityOperations/update

6.16.3 Resource methods

6.16.3.1 POST

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

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

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

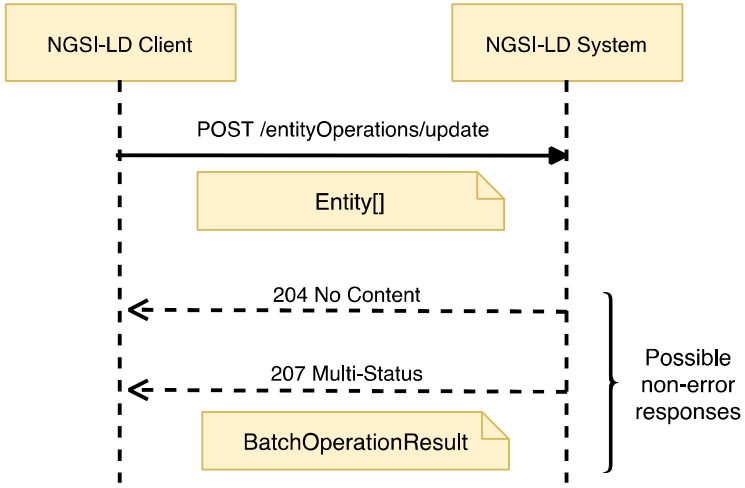

Figure 6.16.3.1-1: Batch Entity Update Interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of Entities to be updated	
Response Body	**Data Type**	**Cardinality**	**Response Code**	**Remarks**
	N/A	N/A	204 No Content	If all entities have been successfully updated, there is no payload body in the response.
	BatchOperationResult	1	207 Multi-Status	If only some or none of the entities have been successfully updated, a response body containing the result of each operation contained in the batch is returned in a BatchOperationResult structure. It contains two arrays. The first array ('success') contains the URIs of the successfully updated entities, while the second array ('errors') contains information about the error for each of the entities that could not be updated. There is no restriction as to the order of the Entity Ids in the arrays. If any of the entities matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.17 Resource: entityOperations/delete

6.17.1 Description

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

6.17.2 Resource definition

Resource URI:

- /entityOperations/delete

6.17.3 Resource methods

6.17.3.1 POST

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

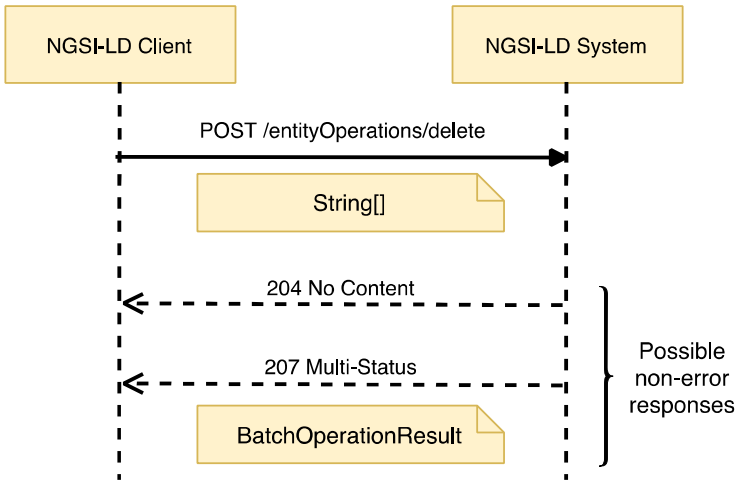

Figure 6.17.3.1-1: Batch Entity Delete Interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	String[]	1	Array of String (URIs representing Entity IDs) to be deleted	
Response Body	**Data Type**	**Cardinality**	**Response Code**	**Remarks**
	N/A	N/A	204 No Content	If all entities existed and have been successfully deleted, there is no payload body in the response.
	BatchOperationResult	1	207 Multi-Status	If some or all of the entities have not been successfully deleted, or did not exist, a response body containing the result of each operation contained in the batch is returned in a BatchOperationResult structure. It contains two arrays. The first array ('success') contains the URIs of the successfully deleted entities, while the second array ('errors') contains information about the error for each of the entities that could not be deleted. There is no restriction as to the order of the Entity Ids in the arrays. If any of the entities matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.

	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.18 Resource: temporal/entities/

6.18.1 Description

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

6.18.2 Resource definition

Resource URI:

- /temporal/entities/

6.18.3 Resource methods

6.18.3.1 POST

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

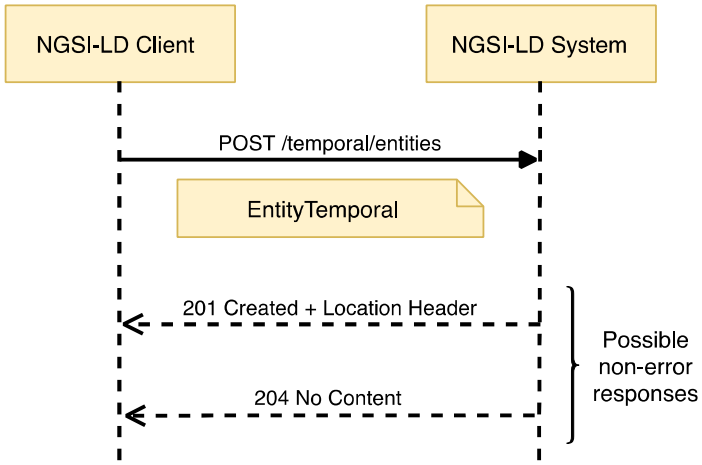

Figure 6.18.3.1-1: Create or Update Temporal Representation of Entity interaction

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	EntityTemporal	1	Payload body in the request contains a JSON-LD object which represents the temporal representation of the entity that is to be created (or updated).	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	Upon creation success, the HTTP response shall include a "Location" HTTP header that contains the resource URI of the created entity resource.
	N/A	N/A	204 No Content	Upon update success.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	422 Unprocessable Entity	It is used to indicate that the operation is not available, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.18.3.2 GET

This method is associated to the operation "Query Temporal Evolution of Entities" and shall exhibit the behaviour defined by clause 5.7.4, providing the temporal evolution of the matching Entities as part of the HTTP response payload body. In addition to this method, an alternative way to perform "Query Temporal Evolution of Entities" operations via POST is defined in clause 6.24. Figure 6.18.3.2-1 shows this interaction.

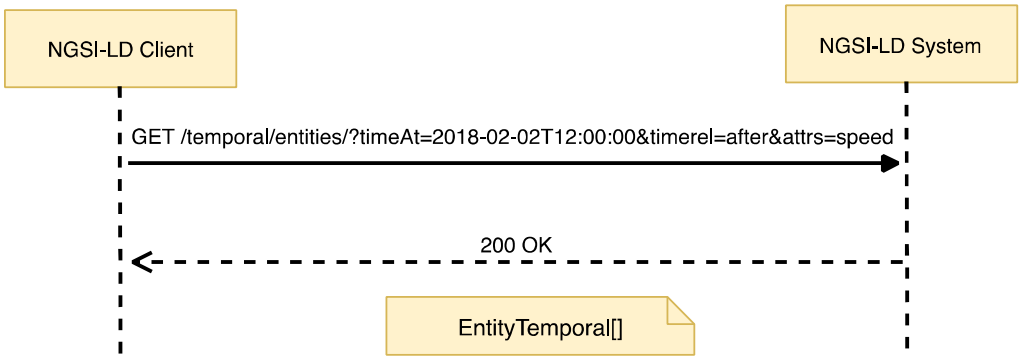

Figure 6.18.3.2-1: Query Temporal Evolution of Entities interaction

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

Table 6.18.3.2-1: Temporal Evolution Query parameters

Name	Data Type	Cardinality	Remarks
id	Comma separated list of strings	0..1	Each String shall be a valid URI. List of entity ids to be retrieved.
type	String	0..1 It shall be 1 if *attrs* is not present	Selection of Entity Types as per clause 4.17.
idPattern	Regular expression as defined by [11]	0..1	Regular expression that shall be matched by entity ids.
attrs	Comma separated list of strings	0..1 It shall be 1 if *type* is not present	Each String is an Attribute (Property or Relationship) name. List of Attributes (Properties or Relationships) to be retrieved.
q	String	0..1	Query as per clause 4.9.
csf	String	0..1	Context Source filter as per clause 4.9.
geometry	String	0..1 It shall be 1 if *georel* or *coordinates* are present	Geometry as per clause 4.10. It is part of geoquery.
georel	String	0..1 It shall be 1 if *geometry* or *coordinates* are present	Geo relationship as per clause 4.10. It is part of geoquery.
coordinates	String	0..1 It shall be one if *georel* or *geometry* are present	Coordinates serialized as a string as per clause 4.10. It is part of geoquery.
geoproperty	String	0..1 It shall be ignored if no geoquery is present	The name of the Property that contains the geospatial data that will be used to resolve the geoquery. By default, will be *location* (see clause 4.7).
timeproperty	String	0..1	It represents a Temporal Property Name. Allowed values: "observedAt", "createdAt", "modifiedAt" and "deletedAt". If not specified, the default is "observedAt" (see clause 4.8).
timerel	String	1	It represents the temporal relationship as defined by clause 4.11. Allowed values: "before", "after", "between".
timeAt	String	1	representing the *timeAt* parameter as defined by clause 4.11. It shall be a *DateTime.*
endTimeAt	String	0..1	It representing the *endTimeAt* parameter as defined by clause 4.11. It shall be a *DateTime.* Cardinality shall be 1 if *timerel* is equal to "between".
lastN	Positive integer	0..1	Only the last n instances, per Attribute, per Entity (under the specified time interval) shall be retrieved.
lang	String	0..1	It represents the preferred natural language of the response. It is used to reduce languageMaps to a string or string array property in a single preferred language.
aggrMethods	Comma separated list of strings	0..1 It shall be 1 if *aggregatedValues* is present in the *options* parameter	Each String represents an aggregation method, as defined by clause 4.5.19. Only applicable if *aggregatedValues* is present in the *options* parameter.

Name	Data Type	Cardinality	Remarks
aggrPeriodDuration	String	0..1	It represents the duration of each period used for the aggregation as defined by clause 4.5.19. If not specified, it defaults to a duration of 0 seconds and is interpreted as a duration spanning the whole time-range specified by the temporal query. Only applicable if *aggregatedValues* is present in the *options* parameter
scopeQ	String	0..1	Scope query (see clause 4.19).

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

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	EntityTemporal[]	1	200 OK	A response body containing the query result as a list of temporal representation of Entities.
	ProblemDetails (see IETF RFC 7807 [10])	1	400 Bad Request	It is used to indicate that the request or its content is incorrect, see clause 6.3.2. In the returned *ProblemDetails* structure, the "detail" attribute should convey more information about the error.

6.19 Resource: temporal/entities/{entityId}

6.19.1 Description

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

6.19.2 Resource definition

Resource URI:

- /temporal/entities/{entityId}

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

Table 6.19.2-1: URI variables

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

6.19.3 Resource methods

6.19.3.1 GET

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

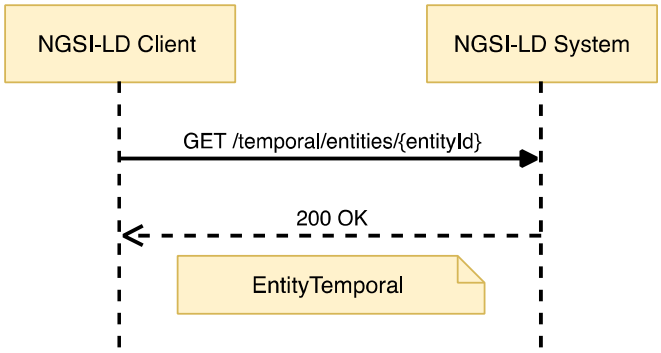

Figure 6.19.3.1-1: Retrieve Temporal evolution of an Entity interaction

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

Table 6.19.3.1-1: Query parameters

Name	Data Type	Cardinality	Remarks
attrs	Comma separated list of strings	0..1	Each String is an Attribute (Property or Relationship) name. List of Attributes to be retrieved. If not specified, all Attributes related to the temporal representation of an entity shall be retrieved.
timeproperty	String	0..1	It represents a Temporal Property Name. Allowed values: "observedAt", "createdAt", "modifiedAt" and "deletedAt". If not specified, the default is "observedAt". (See clause 4.8).
timerel	String	0..1 It shall be 1 if *timeAt* is present	It represents the temporal relationship as defined by clause 4.11. Allowed values: "before", "after", "between".
timeAt	String	0..1 It shall be 1 if *timerel* is present	It represents the *timeAt* parameter as defined by clause 4.11. It shall be a *DateTime*.
endTimeAt	String	0..1 It shall be 1 if *timerel* is equal to "between"	It represents the *endTimeAt* parameter as defined by clause 4.11. It shall be a *DateTime.*
lastN	Positive integer	0..1	Only the last n Attribute instances (under the concerned time interval) shall be retrieved.
lang	String	0..1	It represents the preferred natural language of the response. It is used to reduce languageMaps to a string or string array property in a single preferred language.
aggrMethods	Comma separated list of strings	0..1 It shall be 1 if *aggregatedValues* is present in the *options* parameter	Each String represents the aggregation methods as defined by clause 4.5.19. Only applicable if *aggregatedValues* is present in the *options* parameter.
aggrPeriodDuration	String	0..1	It represents the duration of each period used for the aggregation as defined by clause 4.5.19. If not specified, it defaults to a duration of 0 seconds and is interpreted as a duration spanning the whole time-range specified by the temporal query. Only applicable if *aggregatedValues* is present in the *options* parameter.

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

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	Data Type	Cardinality	Response Codes	Remarks
	EntityTemporal	1	200 OK	A response body containing the JSON-LD temporal representation of the target entity containing the selected Attributes.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.19.3.2 DELETE

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

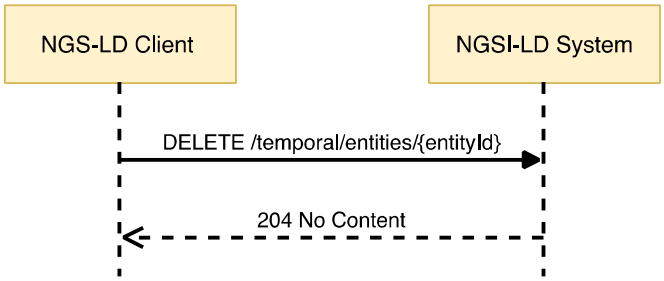

Figure 6.19.3.2-1: Delete Temporal Representation of Entity interaction

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

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	Data Type	Cardinality	Response Codes	Remarks
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

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

6.20.1 Description

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

6.20.2 Resource definition

Resource URI:

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

Resource URI variables for this resource are defined in table 6.20.2-1.

Table 6.20.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity

6.20.3 Resource methods

6.20.3.1 POST

This method is bound to the "Add Attributes to Temporal Representation of an Entity" operation and shall exhibit the behaviour defined by clause 5.6.12. The Entity identifier is the value of the resource URI variable *entityId*. The data to be added shall be contained in the HTTP request payload body. Figure 6.20.3.1-1 shows the Add Attributes interaction and table 6.20.3.1-1 describes the request body and possible responses.

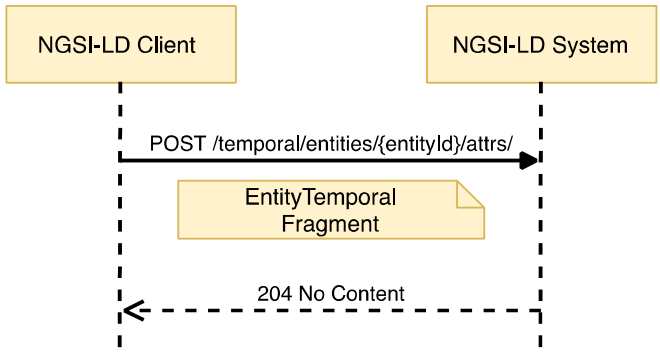

Figure 6.20.3.1-1: Add Attributes to Temporal Representation of an Entity interaction

Table 6.20.3.1-1: Add Attributes to Temporal Representation of an Entity request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	EntityTemporal Fragment	1	EntityTemporal Fragment containing a complete representation of the Attribute instances to be added.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No content	All the Attributes were added successfully.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier (URI) not known to the system, see clause 6.3.2.

6.21 Resource: temporal/entities/{entityId}/attrs/{attrId}

6.21.1 Description

This resource represents an Attribute (Property or Relationship) of a Temporal Representation of an NGSI-LD Entity.

6.21.2 Resource definition

Resource URI:

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

Resource URI variables for this resource are defined in table 6.21.2-1.

Table 6.21.2-1: URI variables

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

6.21.3 Resource methods

6.21.3.1 DELETE

This method is associated to the operation "Delete Attribute from Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.13. The Entity identifier is the value of the resource URI variable *entityId*. The Attribute Name is the value of the resource URI variable *attrId*. Figure 6.21.3.1-1 shows the Delete Attribute from Temporal Representation of an Entity interaction, table 6.21.3.1-1 shows the delete parameters to be supported and table 6.21.3.1-2 describes the request body and possible responses.

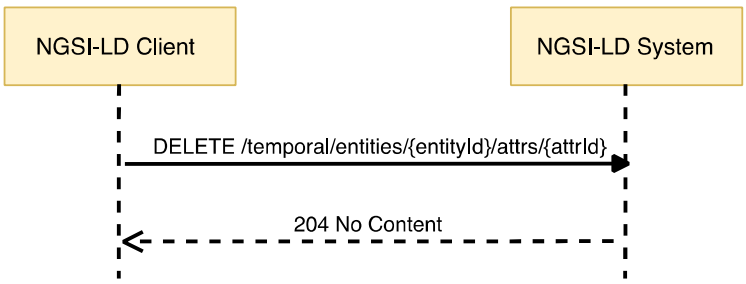

Figure 6.21.3.1-1: Delete Attribute from Temporal Representation of an Entity interaction

Table 6.21.3.1-1: Delete parameters

Name	Data Type	Cardinality	Remarks
deleteAll	Boolean	0..1	If *true,* all attribute instances are deleted. Otherwise (default) only the Attribute instance specified by the *datasetId* is deleted. In case neither the deleteAll flag nor a *datasetId* is present, the default Attribute instance is deleted.
datasetId	String	0..1	Shall be a valid URI. Specifies the *datasetId* of the dataset to be deleted.

Table 6.21.3.1-2: Delete Attribute from Temporal Representation of an Entity request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	Data Type	Cardinality	Response Codes	Remarks
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier (URI) or Attribute Name not known to the system. See clause 6.3.2.

6.22 Resource: temporal/entities/{entityId}/attrs/{attrId}/{instanceId}

6.22.1 Description

This resource represents an Attribute (Property or Relationship) instance of a Temporal Representation of an NGSI-LD Entity.

6.22.2 Resource definition

Resource URI:

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

Resource URI variables for this resource are defined in table 6.22.2-1.

Table 6.22.2-1: URI variables

Name	Definition
entityId	Id (URI) of the concerned entity
attrId	Attribute Name (Property or Relationship)
instanceId	Id (URI) identifying a particular Attribute instance

6.22.3 Resource methods

6.22.3.1 PATCH

This method is associated to the operation "Modify attribute instance from Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.14. The Entity identifier is the value of the resource URI variable *entityId*. The attribute name is the value of the resource URI variable *attrId*. The instance identifier is the value of the resource URI variable *instanceId*. Figure 6.22.3.1-1 shows the Modify Attribute instance interaction and table 6.22.3.1-1 describes the request body and possible responses.

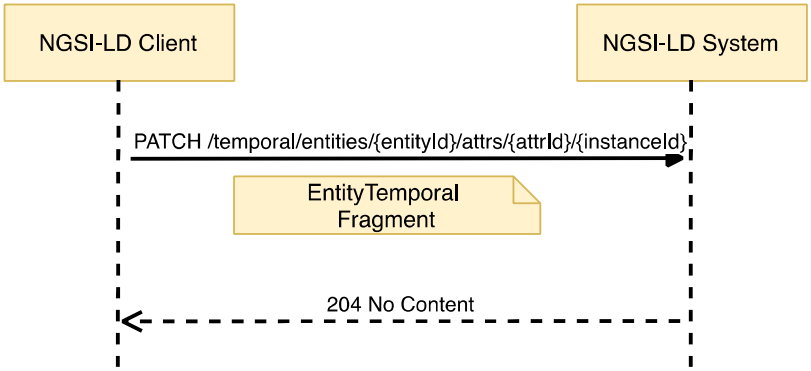

Figure 6.22.3.1-1: Modify Attribute instance from Temporal Representation interaction

Table 6.22.3.1-1: Modify Attribute instance from Temporal Representation request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	EntityTemporal Fragment	1	EntityTemporal Fragment containing a complete representation of the Attribute instance to be replaced.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier (URI), attribute name or instance identifier not known to the system. See clause 6.3.2.

6.22.3.2 DELETE

This method is associated to the operation "Delete Attribute instance from Temporal Representation of an Entity" and shall exhibit the behaviour defined by clause 5.6.15. The Entity identifier is the value of the resource URI variable *entityId*. The Attribute Name is the value of the resource URI variable *attrId*. The instance identifier is the value of the resource URI variable *instanceId*. Figure 6.22.3.2-1 shows the Delete Attribute instance interaction and table 6.22.3.2-1 describes the request body and possible responses.

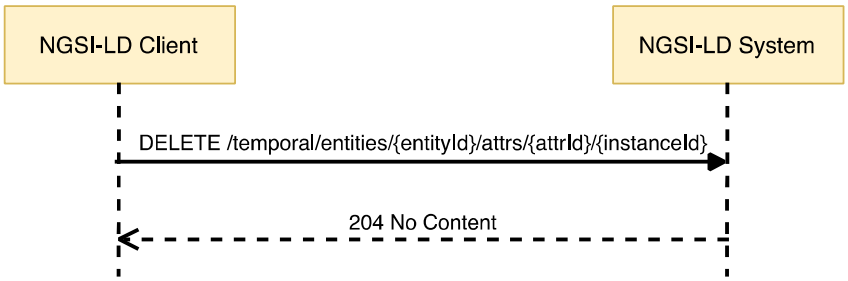

Figure 6.22.3.2-1: Delete Attribute instance from Temporal Representation interaction

Table 6.22.3.2-1: Delete Attribute instance from Temporal Representation request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity identifier (URI), attribute name or instance identifier not known to the system. See clause 6.3.2.

6.23 Resource: entityOperations/query

6.23.1 Description

A sub-resource, pertaining to the *entityOperations/* resource, intended to enable querying for entities by means of a POST method. The behaviour of this clause mirrors the one in clause 6.4.3.2, which performs the "Query Entity" operation (defined by clause 5.7.2) by means of a GET method. The reason to provide an alternative via POST is that, using GET:

a) The client may end up assembling very long URLs, due to the URI parameters for 'id', 'q', type', 'attrs', etc., being included in the URL. Problems with too long URLs may arise with some applications that cut URLs to a maximum length.

b) There is a need to URL-encode the resulting URL. By using POST, there is no need to url-encode.

6.23.2 Resource definition

Resource URI:

- /entityOperations/query

6.23.3 Resource methods

6.23.3.1 POST

This method is associated to the operation "Query Entities" and shall exhibit the behaviour defined by clause 5.7.2. Figure 6.23.3.1-1 shows the operation interaction and table 6.23.3.1-1 describes the request body and possible responses.

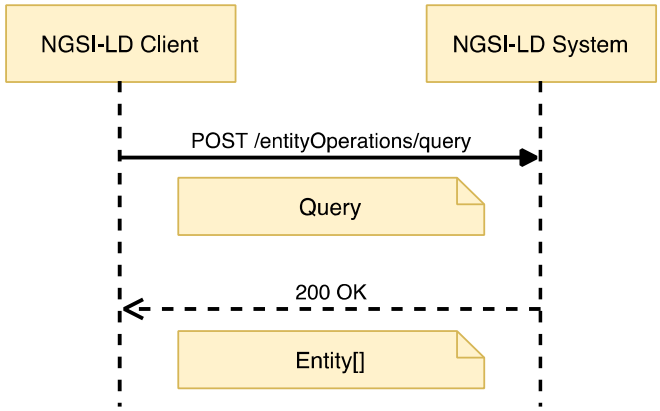

Figure 6.23.3.1-1: Query Entity via POST Interaction

Table 6.23.3.1-1: Query Entity via POST Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Query	1	Payload body in the request contains a JSON-LD object which represents the query to be performed.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Entity[]	1	200 OK	A response body containing the query result as a list of Entities.
	ProblemDetails (see IETF RFC 7807 [10])	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.

6.24 Resource: temporal/entityOperations/query

6.24.1 Description

A sub-resource, pertaining to the *temporal/entityOperations/* resource, intended to enable temporal querying for entities by means of a POST method. The behaviour of this clause mirrors the one in clause 6.18.3.2, which performs the "Query Temporal Evolution of Entities" (defined by clause 5.7.4) operation by means of a GET method. The reason to provide an alternative via POST is that, using GET:

a) The client may end up assembling very long URLs, due to the URI parameters for 'id', 'q', type', 'attrs', etc., being included in the URL. Problems with too long URLs may arise with some applications that cut URLs to a maximum length.

b) There is a need to URL-encode the resulting URL. By using POST, there is no need to url-encode.

6.24.2 Resource definition

Resource URI:

- /temporal/entityOperations/query

6.24.3 Resource methods

6.24.3.1 POST

This method is associated to the operation "Query Temporal Evolution of Entities" and shall exhibit the behaviour defined by clause 5.7.4. Figure 6.24.3.1-1 shows the operation interaction and table 6.24.3.1-1 describes the request body and possible responses.

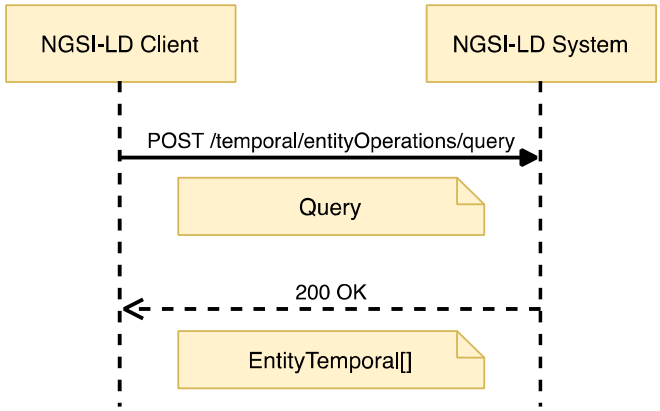

Figure 6.24.3.1-1: Temporal Query Entity via POST Interaction

Table 6.24.3.1-1: Temporal Query Entity via POST Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Query	1	Payload body in the request contains a JSON-LD object which represents the query to be performed.	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	EntityTemporal[]	1	200 OK	A response body containing the query result as a list of Entities.
	ProblemDetails (see IETF RFC 7807 [10])	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.

6.25 Resource: types/

6.25.1 Description

This resource represents the entity types available in an NGSI-LD system.

6.25.2 Resource definition

Resource URI:

- /types/

6.25.3 Resource methods

6.25.3.1 GET

This method is associated to the operations "Retrieve Available Entity Types" and "Retrieve Details of Available Entity Types" (if the "details" parameter is set to true) and shall exhibit the behaviour defined by clauses 5.7.5 and 5.7.6 respectively.

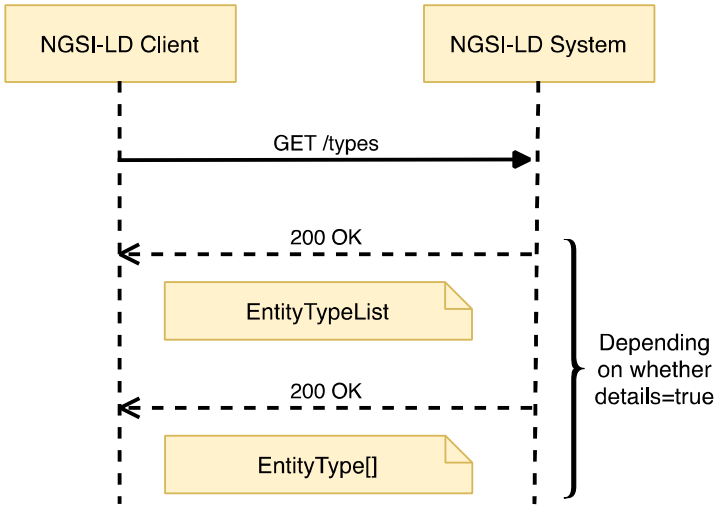

Figure 6.25.3.1-1: Retrieve Available Entity Types interaction

The request parameters that shall be supported are those defined in table 6.25.3.1-1 and table 6.25.3.1-2 describes the request body and possible responses.

Table 6.25.3.1-1: Retrieve Available Entity Types: optional parameter

Name	Data Type	Cardinality	Remarks
details	Boolean	0..1	If *true*, then detailed entity type information represented as an array with elements of the Entity Type data structure (clause 5.2.25) is to be returned

Table 6.25.3.1-2: Retrieve Available Entity Types request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	EntityTypeList	1	200 OK	A response body containing the JSON-LD representation of the EntityTypeList (clause 5.2.24) is to be returned, unless details=true is specified
	EntityType[]	1	200 OK	If details=true is specified, a response body containing a JSON-LD array with elements of the EntityType data structure (clause 5.2.25) is to be returned
	ProblemDetails (see IETF RFC 7807 [10])	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

6.26 Resource: types/{type}

6.26.1 Description

This resource represents the specified entity type for which entity instances are available in an NGSI-LD system.

6.26.2 Resource definition

Resource URI:

- /types/{type}

Resource URI variables for this resource are defined in table 6.26.2-1.

Table 6.26.2-1: URI variables

Name	Definition
type	Name of the entity type for which detailed information is to be retrieved. The Fully Qualified Name (FQN) as well as the short name can be used, given that the latter is part of the JSON-LD @context provided.

6.26.3 Resource methods

6.26.3.1 GET

This method is associated to the operation "Retrieve Available Entity Type Information" and shall exhibit the behaviour defined by clause 5.7.7. The entity type is the value of the resource URI variable "type". Figure 6.26.3.1-1 shows the retrieve available entity type interaction.

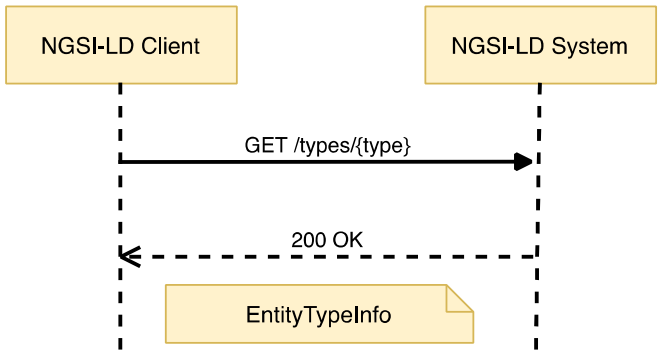

Figure 6.26.3.1-1: Retrieve Available Entity Type interaction

Table 6.26.3.1-1 describes the request body and possible responses.

Table 6.26.3.1-1: Retrieve Available Entity Type request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	EntityTypeInfo	1	200 OK	A response body containing the JSON-LD representation of the detailed information about the available entity type.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an entity type not known to the system, see clause 6.3.2.

6.27 Resource: attributes/

6.27.1 Description

This resource represents the attributes available in an NGSI-LD system.

6.27.2 Resource definition

Resource URI:

- /attributes/

6.27.3 Resource methods

6.27.3.1 GET

This method is associated to the operations "Retrieve Available Attributes" and "Retrieve Details of Available Attributes" (if the "details" parameter is set to true) and shall exhibit the behaviour defined by clauses 5.7.8 and 5.7.9 respectively.

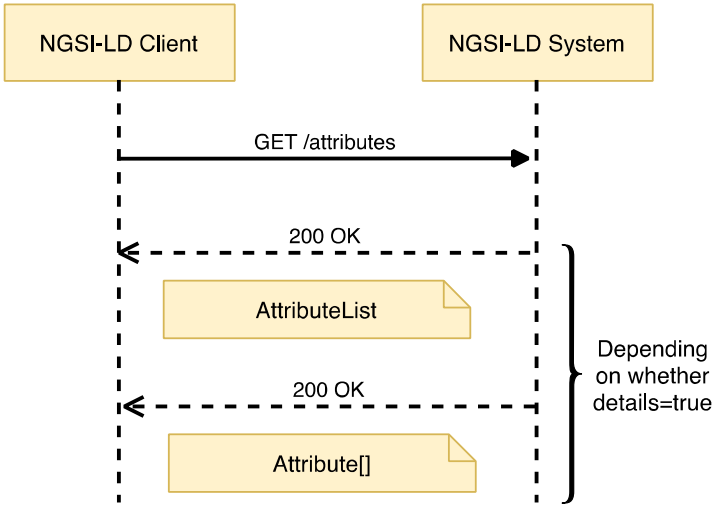

Figure 6.27.3.1-1: Retrieve Available Attributes interaction

The request parameters that shall be supported are those defined in table 6.27.3.1-1 and table 6.27.3.1-2 describes the request body and possible responses.

Table 6.27.3.1-1: Retrieve Available Attributes: optional parameter

Name	Data Type	Cardinality	Remarks
details	Boolean	0..1	If *true*, then detailed attribute information represented as an array with elements of the Attribute data structure (clause 5.2.28) is to be returned

Table 6.27.3.1-2: Retrieve Available Attributes request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	AttributeList	1	200 OK	A response body containing the JSON-LD representation of the AttributeList (clause 5.2.27) is to be returned, unless details=true is specified.
	Attribute[]	1	200 OK	If details=true is specified, a response body containing a JSON-LD array with elements of the Attribute data structure (clause 5.2.28) is to be returned.
	ProblemDetails (see IETF RFC 7807 [10])	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.

6.28 Resource: attributes/{attrId}

6.28.1 Description

This resource represents the specified attribute that belongs to entity instances existing within the NGSI-LD system.

6.28.2 Resource definition

Resource URI:

- /attributes/{attrId}

Resource URI variables for this resource are defined in table 6.28.2-1.

Table 6.28.2-1: URI variables

Name	Definition
attrId	Name of the attribute for which detailed information is to be retrieved. The Fully Qualified Name (FQN) as well as the short name can be used, given that the latter is part of the JSON-LD @context provided.

6.28.3 Resource methods

6.28.3.1 GET

This method is associated to the operation "Retrieve Available Attribute Information" and shall exhibit the behaviour defined by clause 5.7.10. The attribute is the value of the resource URI variable "attrId". Figure 6.28.3.1-1 shows the retrieve available attribute information interaction.

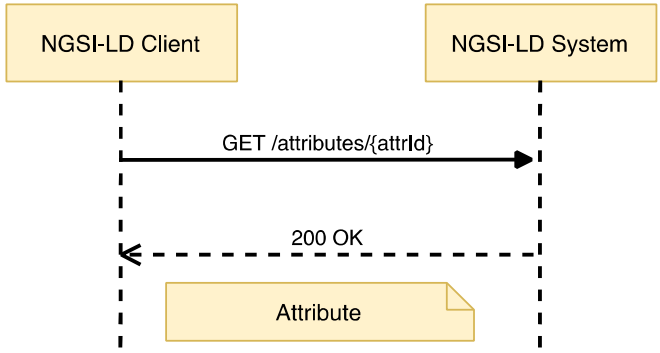

Figure 6.28.3.1-1: Retrieve Available Attribute Information interaction

Table 6.28.3.1-1 describes the request body and possible responses.

Table 6.28.3.1-1: Retrieve Available Attribute Information request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	Attribute	1	200 OK	A response body containing the JSON-LD representation of the detailed information about the available attribute.
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an attribute name not known to the system, see clause 6.3.2.

6.29 Resource: jsonldContexts/

6.29.1 Description

This resource represents the @contexts known to an NGSI-LD system.

6.29.2 Resource definition

Resource URI:

- /jsonldContexts/

6.29.3 Resource methods

6.29.3.1 POST

This method is bound to the operation "Add @context" and shall exhibit the behaviour defined by clause 5.13.2, taking the @context to be added from the HTTP request payload body. Figure 6.29.3.1-1 shows the Add @context interaction and table 6.29.3.1-1 describes the request body and possible responses.

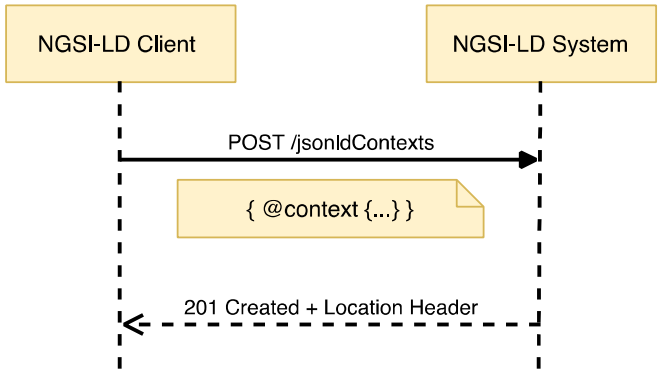

Figure 6.29.3.1-1: Add @context interaction

Table 6.29.3.1-1: Add @context request body and possible responses

	Data Type	Cardinality	Remarks	
Request Body	JSON Object	1	Payload body in the request contains a JSON object that has a root node named @context, which represents a JSON-LD "local context".	
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	201 Created	The HTTP response shall include a "Location" HTTP header that contains the local URI of the added @context.
	ProblemDetails (see IETF RFC 7807 [10])	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.

6.29.3.2 GET

This method is associated to the operation "List @contexts" and shall exhibit the behaviour defined by clause 5.13.3, and it provides information about stored @contexts as part of the HTTP response payload body. Figure 6.29.3.2-1 shows the List @contexts interaction.

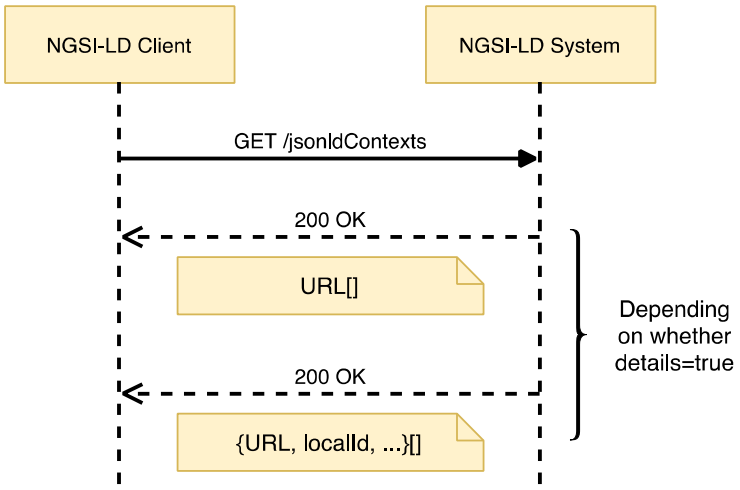

Figure 6.29.3.2-1: List @contexts interaction

The request parameters that shall be supported by implementations are those defined in table 6.29.3.2-1 and table 6.29.3.2-2 describes the request body and possible responses.

Table 6.29.3.2-1: List @contexts request parameters

Name	Data Type	Cardinality	Remarks
details	Boolean	0..1	Whether a list of URLs or a more detailed list of JSON Objects is requested
kind	String	0..1	Can be either "Cached", "Hosted", or "ImplicitlyCreated"

Table 6.29.3.2-2: List @contexts request body and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	String[] or JSON Object[]	1	200 OK	A response body containing a list of URLs or a list of JSON Objects, as defined in clause 5.13.3.5, representing metadata about stored @contexts.
	ProblemDetails (see IETF RFC 7807 [10])	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.

6.30 Resource: jsonldContexts/{contextId}

6.30.1 Description

This resource represents a JSON-LD @context stored in the Broker's internal @context storage.

6.30.2 Resource definition

Resource URI:

- /jsonldContexts/{contextId}

Resource URI variables for this resource are defined in table 6.30.2-1.

Table 6.30.2-1: URI variables

Name	Definition
contextId	Local identifier of the @context to be managed (served or deleted). For @contexts of kind "Cached" this can also be the original URL the Broker downloaded the @context from.

6.30.3 Resource methods

6.30.3.1 GET

This method is associated to the operation "Serve @context" and shall exhibit the behaviour defined by clause 5.13.4. The @context identifier is the value of the resource URI variable "contextId". Figure 6.30.3.1-1 shows the HTTP Serve @context interaction.

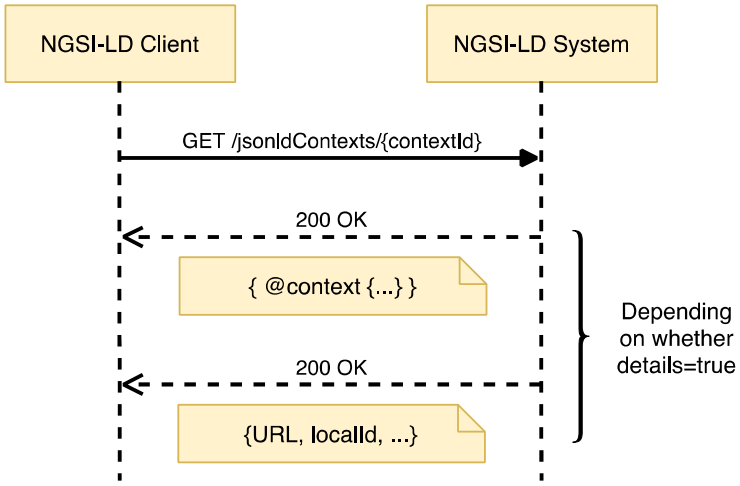

Figure 6.30.3.1-1: Serve @context interaction

The request parameters that shall be supported by implementations are those defined in table 6.30.3.1-1 and table 6.30.3.1-2 describes the request body and possible responses.

Table 6.30.3.1-1: Serve @contexts request parameters

Name	Data Type	Cardinality	Remarks
details	Boolean	0..1	Whether the content of the @context or its metadata is requested

Table 6.30.3.1-2: Serve @context request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	JSON Object	1	200 OK	If the parameter details is False or missing, response body contains a JSON object that has a root node named @context, which represents a JSON-LD "local context". If the parameter details is True, response body contains a JSON object as defined in clause 5.13.4.5, which metadata of a JSON-LD "local context".
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an @context identifier not known to the system, see clause 6.3.2.
	ProblemDetails (see IETF RFC 7807 [10])	1	422 Unprocessable	It is used when a client indicated an @context of type "Cached", see clause 6.3.2.

6.30.3.2 DELETE

This method is associated to the operation "Delete and Reload @context" and shall exhibit the behaviour defined by clause 5.13.5. The entity identifier is the value of the resource URI variable "contextId". Figure 6.30.3.2-1 shows the delete entity interaction. The request parameters that shall be supported are those defined in table 6.30.3.2-1 and table 6.30.3.2-2 describes the request body and possible responses.

Table 6.30.3.2-1: Delete and Reload @context request parameters

Name	Data Type	Cardinality	Remarks
reload	Boolean	0..1	indicates to perform a download and replace of the @context, as specified in clause 5.13.5.4.

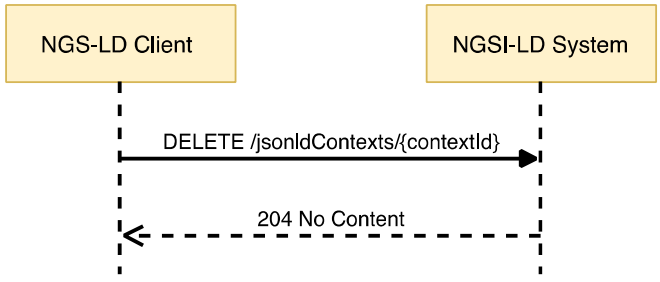

Figure 6.30.3.2-1: Delete and Reload @context interaction

Table 6.30.3.2-2: Delete and Reload @context request body and possible responses

Request Body	Data Type	Cardinality	Remarks	
	N/A	N/A		
Response Body	**Data Type**	**Cardinality**	**Response Codes**	**Remarks**
	N/A	N/A	204 No Content	
	ProblemDetails (see IETF RFC 7807 [10])	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 (see IETF RFC 7807 [10])	1	404 Not Found	It is used when a client provided an @context identifier not known to the system, see clause 6.3.2.
	ProblemDetails (see IETF RFC 7807 [10])	1	503 Service Unavailable	It is used when re-downloading fails.

6.31 Resource: entityOperations/merge

6.31.1 Description

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

6.31.2 Resource definition

Resource URI:

- /entityOperations/merge

6.31.3 Resource methods

6.31.3.1 POST

This method is associated to the operation "Batch Entity Merge" and shall exhibit the behaviour defined by clause 5.6.20. Figure 6.31.3.1-1 shows the operation interaction and table 6.31.3.1-1 describes the request body and possible responses.

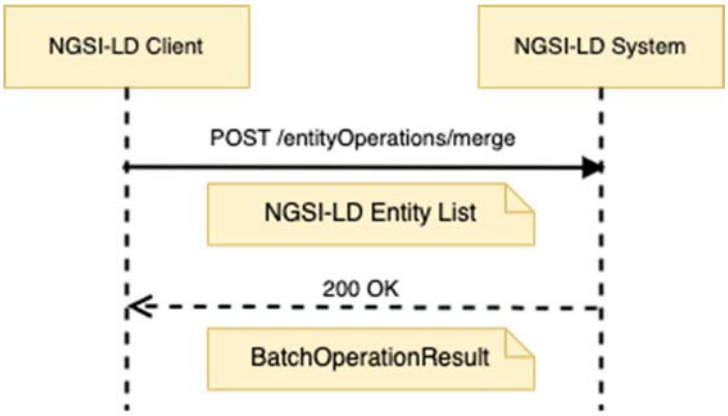

Figure 6.31.3.1-1: Batch Entity Merge Interaction

Table 6.31.3.1-1: Batch Entity Merge Interaction and possible responses

Request Body	**Data Type**	**Cardinality**	**Remarks**	
	Entity[]	1	Array of Entities to be merged.	
Response Body	**Data Type**	**Cardinality**	**Response Code**	**Remarks**
	N/A	N/A	204 No Content	If all entities have been successfully merged, there is no payload body in the response.
	BatchOperationResult	1	207 Multi-Status	If only some or none of the entities have been successfully merged, a response body containing the result of each operation contained in the batch is returned in a BatchOperationResult structure. It contains two arrays. The first array ('success') contains the URIs of the successfully merged entities, while the second array ('errors') contains information about the error for each of the entities that could not be merged-patched. There is no restriction as to the order of the Entity Ids in the arrays. If any of the entities matches to a registration, the relevant parts of the request are forwarded as a distributed operation. In the case when an error response is received back from any distributed operation, a response body containing the result returned from each registration is returned in a BatchOperationResult structure. Errors can occur whenever a distributed operation is unsupported, fails or times out, see clause 6.3.17.
	ProblemDetails (see IETF RFC 7807 [10])	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.

7 API MQTT Notification Binding

7.1 Introduction

This clause defines the optional support of the NGSI-LD API for sending notifications via the MQTT protocol [24] and [25]. The subscriptions are handled using the HTTP binding as described in clause 6, but instead of an HTTP endpoint, an MQTT endpoint is provided.

7.2 Notification behaviour

In case a subscription received via HTTP specifies an MQTT endpoint in the "notification.endpoint.uri" member of the subscription structure (defined by clauses 5.2.12, 5.2.14 and 5.2.15), and the MQTT notification binding is supported by the NGSI-LD implementation, notifications related to this subscription shall be sent via the MQTT protocol.

The syntax of an MQTT endpoint URI is mqtt[s]://[<username>][:<password>]@<host>[:<port>]/<topic>[/<subtopic>]* and follows an existing convention for representing an MQTT endpoint as a URI [i.19].

Username and password can be optionally specified as part of the endpoint URI. If the port is not explicitly specified, the default MQTT port is 1883 for MQTT over TCP and 8883 for mqtts, i.e. Secure MQTT over TLS. MQTT supports the structuring of topics as a hierarchy with any number of subtopic levels, which can be specified as part of the endpoint URI.

In MQTT, all non-protocol information has to be included into the MQTT message. This means that the actual notification as specified in clause 5.3.1, as well as additional information like MIME type, possibly the link to the @context and additional user-specified information, which in the HTTP case is provided as headers, has to be included into the MQTT message. The MQTT notification message shall be provided as a JSON Object with the two elements "metadata" and "body". The actual notification, as specified in clause 5.3.1 is the value of "body", whereas any additional information is provided as key-value pairs in "metadata".

For the MQTT protocol, there are currently two versions supported, MQTTv3.1.1 [24] and MQTTv5.0 [25]. Also, there are three levels of quality of service:

- at most once (0);
- at least once (1); and
- exactly once (2).

These can be specified in the subscription as part of the optional array of KeyValuePair type (defined by clause 5.2.22) "notification.endpoint.notifierInfo". The MQTT protocol parameters can be found in table 7.2-1. If not present, the given default value is used.

Table 7.2-1: Protocol parameters for MQTT in notifierInfo

Key	Possible Values	Default	Source	Description
MQTT-Version	mqtt3.1.1, mqtt5.0	mqtt5.0	Subscription's notification.endpoint.notifierInfo	Version of MQTT protocol
MQTT-QoS	0, 1, 2	0	Subscription's notification.endpoint.notifierInfo	MQTT Quality of service, at most once (0), at least once (1) and exactly once (2)

The MIME type associated with the notification shall be "application/json" by default. However, this can be changed to application/ld+json by means of the "endpoint.accept" member. The MIME type is specified as Content-Type in the "metadata" element of the MQTT message. If the target MIME type is "application/json" then the reference to the JSON-LD @context is provided as Link in the "metadata" element of the MQTT message, following the specification of the HTTP Link header as mandated by the JSON-LD specification [2], section 6.2 (to the default JSON-LD @context if none available). Table 7.2-2 lists these "receiver side" metadata parameters.

Table 7.2-2: Parameters for MQTT in "metadata"

Key	Possible Values	Default	Source	Description
Content-Type	application/json, application/ld+json	application/json	Subscription's notification.endpoint.accept	MIME type of the notification included in the "body" element of the MQTT message
Link	Same format as specified in JSON-LD specification [2], section 6.2 for the HTTP Link header		Link Header provided in Subscription	Contains the reference to the @context in case Content-Type is application/json. Example: <http://myhost.org/mycontext>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

Additionally, if the optional array of KeyValuePair type (defined by clause 5.2.22) "notification.endpoint.receiverInfo" of the subscription is present, then a new entry for each member named "key" of the key, value pairs that make up the array shall be generated and added to the "metadata" element of the MQTT message. The content of each entry shall be set equal to the content of the corresponding "value" member of the KeyValuePair.

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 is the definition of the Core NGSI-LD @context which shall be supported by implementations.

Such definition has been tested using [i.7].


```
{
  "@context": {
    "ngsi-ld": "https://uri.etsi.org/ngsi-ld/",
    "geojson": "https://purl.org/geojson/vocab#",
    "id": "@id",
    "type": "@type",
    "Attribute": "ngsi-ld:Attribute",
    "AttributeList": "ngsi-ld:AttributeList",
    "ContextSourceNotification": "ngsi-ld:ContextSourceNotification",
    "ContextSourceRegistration": "ngsi-ld:ContextSourceRegistration",
    "Date": "ngsi-ld:Date",
    "DateTime": "ngsi-ld:DateTime",
    "EntityType": "ngsi-ld:EntityType",
    "EntityTypeInfo": "ngsi-ld:EntityTypeInfo",
    "EntityTypeList": "ngsi-ld:EntityTypeList",
    "Feature": "geojson:Feature",
    "FeatureCollection": "geojson:FeatureCollection",
    "GeoProperty": "ngsi-ld:GeoProperty",
    "GeometryCollection": "geojson:GeometryCollection",
    "LineString": "geojson:LineString",
    "LanguageProperty": "ngsi-ld:LanguageProperty",
    "MultiLineString": "geojson:MultiLineString",
    "MultiPoint": "geojson:MultiPoint",
    "MultiPolygon": "geojson:MultiPolygon",
    "Notification": "ngsi-ld:Notification",
    "Point": "geojson:Point",
    "Polygon": "geojson:Polygon",
    "Property": "ngsi-ld:Property",
    "Relationship": "ngsi-ld:Relationship",
    "Subscription": "ngsi-ld:Subscription",
    "TemporalProperty": "ngsi-ld:TemporalProperty",
    "Time": "ngsi-ld:Time",
    "VocabularyProperty": "ngsi-ld:VocabularyProperty",
    "accept": "ngsi-ld:accept",
    "attributeCount": "attributeCount",
    "attributeDetails": "attributeDetails",
    "attributeList": {
      "@id": "ngsi-ld:attributeList",
      "@type": "@vocab"
    },
    "attributeName": {
      "@id": "ngsi-ld:attributeName",
      "@type": "@vocab"
    },
    "attributeNames": {
      "@id": "ngsi-ld:attributeNames",
      "@type": "@vocab"
    },
    "attributeTypes": {
      "@id": "ngsi-ld:attributeTypes",
      "@type": "@vocab"
    },
    "attributes": {
      "@id": "ngsi-ld:attributes",
      "@type": "@vocab"
    },
    "attrs": "ngsi-ld:attrs",
    "avg": {
      "@id": "ngsi-ld:avg",
      "@container": "@list"
    },
    "bbox": {
      "@container": "@list",
      "@id": "geojson:bbox"
    },
    "cacheDuration": "ngsi-ld:cacheDuration",
    "contextSourceInfo": "ngsi-ld:contextSourceInfo",
    "cooldown": "ngsi-ld:cooldown",
```

```
"coordinates": {
  "@container": "@list",
  "@id": "geojson:coordinates"
},
"createdAt": {
  "@id": "ngsi-ld:createdAt",
  "@type": "DateTime"
},
"csf": "ngsi-ld:csf",
"data": "ngsi-ld:data",
"dataset": {
  "@id": "ngsi-ld:hasDataset",
  "@container": "@index"
},
"datasetId": {
  "@id": "ngsi-ld:datasetId",
  "@type": "@id"
},
"deletedAt": {
  "@id": "ngsi-ld:deletedAt",
  "@type": "DateTime"
},
"description": "http://purl.org/dc/terms/description",
"detail": "ngsi-ld:detail",
"distinctCount": {
  "@id": "ngsi-ld:distinctCount",
  "@container": "@list"
},
"endAt": {
  "@id": "ngsi-ld:endAt",
  "@type": "DateTime"
},
"endTimeAt": {
  "@id": "ngsi-ld:endTimeAt",
  "@type": "DateTime"
},
"endpoint": "ngsi-ld:endpoint",
"entities": "ngsi-ld:entities",
"entityCount": "ngsi-ld:entityCount",
"entityId": {
  "@id": "ngsi-ld:entityId",
  "@type": "@id"
},
"error": "ngsi-ld:error",
"errors": "ngsi-ld:errors",
"expiresAt": {
  "@id": "ngsi-ld:expiresAt",
  "@type": "DateTime"
},
"features": {
  "@container": "@set",
  "@id": "geojson:features"
},
"format": "ngsi-ld:format",
"geoQ": "ngsi-ld:geoQ",
"geometry": "geojson:geometry",
"geoproperty": "ngsi-ld:geoproperty",
"georel": "ngsi-ld:georel",
"idPattern": "ngsi-ld:idPattern",
"information": "ngsi-ld:information",
"instanceId": {
  "@id": "ngsi-ld:instanceId",
  "@type": "@id"
},
"isActive": "ngsi-ld:isActive",
"key": "ngsi-ld:hasKey",
"lang": "ngsi-ld:lang",
"languageMap": {
  "@id": "ngsi-ld:hasLanguageMap",
  "@container": "@language"
},
"languageMaps": {
  "@id": "ngsi-ld:hasLanguageMaps",
  "@container": "@list"
},
"lastFailure": {
  "@id": "ngsi-ld:lastFailure",
  "@type": "DateTime"
```

```
},
"lastNotification": {
  "@id": "ngsi-ld:lastNotification",
  "@type": "DateTime"
},
"lastSuccess": {
  "@id": "ngsi-ld:lastSuccess",
  "@type": "DateTime"
},
"localOnly": "ngsi-ld:localOnly",
"location": "ngsi-ld:location",
"management": "ngsi-ld:management",
"managementInterval": "ngsi-ld:managementInterval",
"max": {
  "@id": "ngsi-ld:max",
  "@container": "@list"
},
"min": {
  "@id": "ngsi-ld:min",
  "@container": "@list"
},
"mode": "ngsi-ld:mode",
"modifiedAt": {
  "@id": "ngsi-ld:modifiedAt",
  "@type": "DateTime"
},
"notification": "ngsi-ld:notification",
"notificationTrigger": "ngsi-ld:notificationTrigger",
"notifiedAt": {
  "@id": "ngsi-ld:notifiedAt",
  "@type": "DateTime"
},
"notifierInfo": "ngsi-ld:notifierInfo",
"notUpdated": "ngsi-ld:notUpdated",
"object": {
  "@id": "ngsi-ld:hasObject",
  "@type": "@id"
},
"objects": {
  "@id": "ngsi-ld:hasObjects",
  "@container": "@list"
},
"observationInterval": "ngsi-ld:observationInterval",
"observationSpace": "ngsi-ld:observationSpace",
"observedAt": {
  "@id": "ngsi-ld:observedAt",
  "@type": "DateTime"
},
"operationSpace": "ngsi-ld:operationSpace",
"operations": "ngsi-ld:operations",
"previousLanguageMap": {
  "@id": "ngsi-ld:hasPreviousLanguageMap",
  "@container": "@language"
},
"previousObject": {
  "@id": "ngsi-ld:hasPreviousObject",
  "@type": "@id"
},
"previousValue": "ngsi-ld:hasPreviousValue",
"previousVocab": {
  "@id": "ngsi-ld:hasPreviousVocab",
  "@type": "@vocab"
},
"properties": "geojson:properties",
"propertyNames": {
  "@id": "ngsi-ld:propertyNames",
  "@type": "@vocab"
},
"q": "ngsi-ld:q",
"reason": "ngsi-ld:reason",
"receiverInfo": "ngsi-ld:receiverInfo",
"refreshRate": "ngsi-ld:refreshRate",
"registrationId": "ngsi-ld:registrationId",
"registrationName": "ngsi-ld:registrationName",
"relationshipNames": {
  "@id": "ngsi-ld:relationshipNames",
  "@type": "@vocab"
},
```

```
"scope": "ngsi-ld:scope",
"scopeQ": "ngsi-ld:scopeQ",
"showChanges": "ngsi-ld:showChanges",
"startAt": {
  "@id": "ngsi-ld:startAt",
  "@type": "DateTime"
},
"status": "ngsi-ld:status",
"stddev": {
  "@id": "ngsi-ld:stddev",
  "@container": "@list"
},
"subscriptionId": {
  "@id": "ngsi-ld:subscriptionId",
  "@type": "@id"
},
"subscriptionName": "ngsi-ld:subscriptionName",
"success": {
  "@id": "ngsi-ld:success",
  "@type": "@id"
},
"sum": {
  "@id": "ngsi-ld:sum",
  "@container": "@list"
},
"sumsq": {
  "@id": "ngsi-ld:sumsq",
  "@container": "@list"
},
"sysAttrs": "ngsi-ld:sysAttrs",
"temporalQ": "ngsi-ld:temporalQ",
"tenant": {
  "@id": "ngsi-ld:tenant",
  "@type": "@id"
},
"throttling": "ngsi-ld:throttling",
"timeAt": {
  "@id": "ngsi-ld:timeAt",
  "@type": "DateTime"
},
"timeInterval": "ngsi-ld:timeInterval",
"timeout": "ngsi-ld:timeout",
"timeproperty": "ngsi-ld:timeproperty",
"timerel": "ngsi-ld:timerel",
"timesFailed": "ngsi-ld:timesFailed",
"timesSent": "ngsi-ld:timesSent",
"title": "http://purl.org/dc/terms/title",
"totalCount": {
  "@id": "ngsi-ld:totalCount",
  "@container": "@list"
},
"triggerReason": "ngsi-ld:triggerReason",
"typeList": {
  "@id": "ngsi-ld:typeList",
  "@type": "@vocab"
},
"typeName": {
  "@id": "ngsi-ld:typeName",
  "@type": "@vocab"
},
"typeNames": {
  "@id": "ngsi-ld:typeNames",
  "@type": "@vocab"
},
"unchanged": "ngsi-ld:unchanged",
"unitCode": "ngsi-ld:unitCode",
"updated": "ngsi-ld:updated",
"uri": "ngsi-ld:uri",
"value": "ngsi-ld:hasValue",
"values": {
  "@id": "ngsi-ld:hasValues",
  "@container": "@list"
},
"vocab": {
  "@id": "ngsi-ld:hasVocab",
  "@type": "@vocab"
},
"vocabs": {
```

```
      "@id": "ngsi-ld:hasVocabs",
      "@container": "@list"
    },
    "watchedAttributes": {
      "@id": "ngsi-ld:watchedAttributes",
      "@type": "@vocab"
    },
    "@vocab": "https://uri.etsi.org/ngsi-ld/default-context/"
  }
}
```

NOTE: Implementers can take advantage of prefixed terms, i.e. in the form ngsi-ld:term, 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.

JSON representations of the examples shown in this annex can be found at [i.15].

C.2 Entity Representation

C.2.1 Property Graph

Figure C.2.1-1 shows a diagram representing a property graph to be used for the examples discussed in this clause.

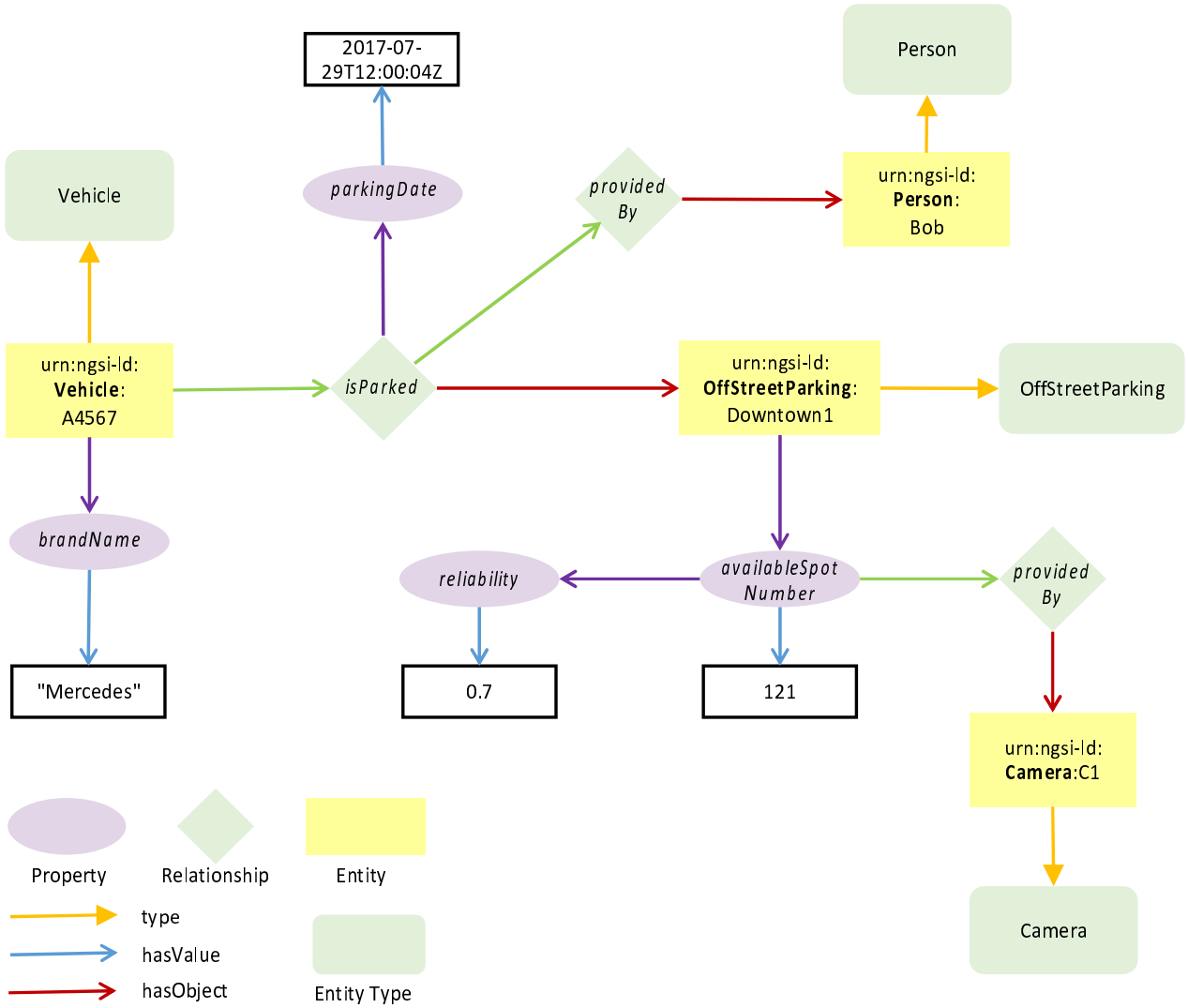

Figure C.2.1-1: Reference example

As per the algorithms described above and as per the rules for generating the JSON-LD representation of NGSI-LD entities the above graph will result in the following JSON-LD representations. The syntax has been checked using the JSON-LD Playground tool [i.5].

C.2.2 Vehicle Entity

Normalized Representation

The normalized representation is a lossless representation of an Entity, where every **Property** is defined by a "type" and a "value" and every **Relationship** is defined by a "type" and an "object".

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"
    },
    "street": {
        "type": "LanguageProperty",
        "languageMap": {
            "fr": "Grand Place",
            "nl": "Grote Markt
        }
    },
    "isParked": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:OffStreetParking:Downtown1",
        "observedAt": "2017-07-29T12:00:04Z",
        "providedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Person:Bob"
        }
    },
    "category": {
        "type": "VocabularyProperty",
        "vocab": "non-commercial"
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Normalized Representation when Language Filter is used

When the Language Filter (see clause 4.15) is used, Properties of type "LanguageProperty" are returned as type "Property", and their languageMaps are reduced to simple strings. For example if the language filter `lang=fr` is specified, only the value for French language is present.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": {
        "type": "Property",
        "value": "Mercedes"
    },
    "street": {
        "type": "Property",
        "value": "Grand Place",
        "lang": "fr"
    },
    "isParked": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:OffStreetParking:Downtown1",
        "observedAt": "2017-07-29T12:00:04Z",
        "providedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Person:Bob"
        }
```

```
    },
    "category": {
        "type": "VocabularyProperty",
        "vocab": "non-commercial"
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Concise Representation

The concise representation is a terser, lossless form of the normalized representation, where redundant Attribute "type" members are omitted and the following rules are applied:

- Every **Property** without further sub-attributes is represented directly by the **Property** value only.
- Every **Property** that includes further sub-attributes is represented by a value key-value pair.
- Every **GeoProperty** without further sub-attributes is represented by the **GeoProperty's** GeoJSON representation only.
- Every **GeoProperty** that includes further sub-attributes is represented by a value key-value pair.
- Every **LanguageProperty** is represented by a languageMap key-value pair.
- Every **VocabularyProperty** is represented by a vocab the value of which is a compacted URI.
- Every **Relationship** is represented by an object key-value pair.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": "Mercedes",
    "street": {
        "languageMap": {
            "fr": "Grand Place",
            "nl": "Grote Markt"
        }
    },
    "isParked": {
        "object": "urn:ngsi-ld:OffStreetParking:Downtown1",
        "observedAt": "2017-07-29T12:00:04Z",
        "providedBy": {
            "object": "urn:ngsi-ld:Person:Bob"
        }
    },
    "category": {
        "vocab": "non-commercial"
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Concise representation when Language Filter is used

The rules apply as defined in the previous examples. For example if the language filter `lang=fr` is specified.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": "Mercedes"
    },
    "street": {
        "value": "Grand Place",
        "lang": "fr"
    },
```

```
    "isParked": {
        "object": "urn:ngsi-ld:OffStreetParking:Downtown1",
        "observedAt": "2017-07-29T12:00:04Z",
        "providedBy": {
            "object": "urn:ngsi-ld:Person:Bob"
        }
    },
    "category": {
        "vocab": "non-commercial"
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Simplified Representation

The simplified representation is a collapsed, lossy representation of an Entity, which focuses on Property Values and Relationship objects present at the first level of the graph only.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": "Mercedes",
    "street": {
        "languageMap": {
            "fr": "Grand Place",
            "nl": "Grote Markt"
        }
    }
    "isParked": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "category": {
        "vocab": "non-commercial"
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Simplified Representation of Natural Language Attributes

The simplified natural language representation is a collapsed representation of an Entity, which focuses on Property Values and Relationship objects present at the first level of the graph, and where languageMaps are reduced to simple string properties. For example if the language filter `lang=fr` is specified.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "brandName": "Mercedes",
    "street": "Grand Place",
    "isParked": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "category": {
        "vocab": "non-commercial"
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Multiple attribute example

Below is an example, where the speed of the car is provided by two different sources. As both may be relevant at the same time, there are two individual attribute instances for speed; each is identified by a *datasetId* and both instances are represented in an array. The *datasetId* enables individually creating, updating and deleting a particular instance without affecting the instance from another source.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "speed": [{
        "type": "Property",
        "value": 55,
        "source": {
            "type": "Property",
            "value": "Speedometer"
        },
        "datasetId": "urn:ngsi-ld:Property:speedometerA4567-speed"
    },
    {
        "type": "Property",
        "value": 54.5,
        "source": {
            "type": "Property",
            "value": "GPS"
        },
        "datasetId": "urn:ngsi-ld:Property:gpsBxyz123-speed"
    }],
    "@context": [
        {
            "Vehicle": "http://example.org/Vehicle",
            "speed": "http://example.org/speed",
            "source": "http://example.org/hasSource"
        },
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Simplified Representation of a multi-attribute

The simplified representation is a collapsed, lossy representation of an Entity, which focuses on Property Values and Relationship objects present at the first level of the graph only.

```
{
    "id": "urn:ngsi-ld:Vehicle:A4567",
    "type": "Vehicle",
    "speed": {
        "dataset": {
            "urn:ngsi-ld:Property:speedometerA4567-speed": 55,
            "urn:ngsi-ld:Property:gpsBxyz123-speed": 54.5
        }
    },
    "@context": [
        {
            "Vehicle": "http://example.org/Vehicle",
            "speed": "http://example.org/speed"
        },
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

C.2.3 Parking Entity

Normalized Representation

The normalized representation is a lossless representation of an Entity, where every **Property** is defined by a "type" and a "value" and every **Relationship** is defined by a "type" and an "object".

Below there is a representation of an Entity of Type "OffStreetParking". It can be observed that the @context is composed of two different elements, the Core one and the vocabulary-specific one.

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "OffStreetParking",
    "name": {
        "type": "Property",
        "value": "Downtown One"
    },
    "availableSpotNumber": {
        "type": "Property",
        "value": 121,
```

```
        "observedAt": "2017-07-29T12:05:02Z",
        "reliability": {
            "type": "Property",
            "value": 0.7
        },
        "providedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Camera:C1"
        }
    },
    "totalSpotNumber": {
        "type": "Property",
        "value": 200
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [-8.5, 41.2]
        }
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Concise Representation

The concise representation is a terser, lossless form of the normalized representation, where redundant Attribute "type" members are omitted and the following rules are applied:

- Every **Property** without further sub-attributes is represented by the **Property** value only.
- Every **Property** that includes further sub-attributes is represented by a value key-value pair.
- Every **GeoProperty** without further sub-attributes is represented by the **GeoProperty's** GeoJSON representation only.
- Every **GeoProperty** that includes further sub-attributes is represented by a value key-value pair.
- Every **LanguageProperty** is defined by a languageMap key-value pair.
- Every **VocabularyProperty** is represented by a vocab the value of which is a compacted URI.
- Every **Relationship** is defined by an object key-value pair.

```
{
  "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
  "type": "OffStreetParking",
  "name": "Downtown One",
  "availableSpotNumber": {
    "value": 121,
    "observedAt": "2017-07-29T12:05:02Z",
    "reliability": 0.7,
    "providedBy": {
      "object": "urn:ngsi-ld:Camera:C1"
    }
  },
  "totalSpotNumber": 200,
  "location": {
    "type": "Point",
    "coordinates": [
      -8.5,
      41.2
    ]
  },
  "@context": [
    "http://example.org/ngsi-ld/latest/parking.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

Simplified representation

The Simplified Representation (also known as keyValues) is a lossy, collapsed representation of an Entity, which focuses on Property Values and Relationship objects present at the first level of the graph only.

```
{
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "OffStreetParking",
    "name": "Downtown One",
    "availableSpotNumber": 121,
    "totalSpotNumber": 200,
    "location": {
        "type": "Point",
        "coordinates": [-8.5, 41.2]
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Normalized GeoJSON Representation

The normalized GeoJSON representation of a single Entity is defined as a single GeoJSON Feature object as follows:

```
{
  "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [-8.51, 41.1]
  },
  "properties": {
    "type": "OffStreetParking",
    "name": {
      "type": "Property",
      "value": "Downtown One"
    },
    "availableSpotNumber": {
      "type": "Property",
      "value": 121,
      "observedAt": "2017-07-29T12:05:02Z",
      "reliability": {
        "type": "Property",
        "value": 0.7
      },
      "providedBy": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:Camera:C1"
      }
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [-8.51, 41.1]
        }
    },
    "totalSpotNumber": {
      "type": "Property",
      "value": 200
    },
    "@context": [
      "http://example.org/ngsi-ld/latest/parking.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
}
```

The GeoJSON representation of multiple Entities is defined as a GeoJSON FeatureCollection object containing an array of GeoJSON features corresponding to the individual Entity representations.

```
{
  "type": "FeatureCollection",
  "features": [
    {
```

```
    "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [-8.5, 41.1]
    },
    "properties": {
      "type": "OffStreetParking",
      "name": {
        "type": "Property",
        "value": "Downtown One"
      },
      "availableSpotNumber": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z",
        "reliability": {
          "type": "Property",
          "value": 0.7
        },
        "providedBy": {
          "type": "Relationship",
          "object": "urn:ngsi-ld:Camera:C1"
        }
      },
      "totalSpotNumber": {
        "type": "Property",
        "value": 200
      },
      "location": {
        "type": "GeoProperty",
        "value": {
          "type": "Point",
          "coordinates": [-8.51, 41.1]
        }
      }
    }
  },
  {
    "id": "urn:ngsi-ld:OffStreetParking:Downtown2",
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [-8.51, 41.1]
    },
    "properties": {
      "type": "OffStreetParking",
      "name": {
        "type": "Property",
        "value": "Downtown Two"
      },
      "availableSpotNumber": {
        "type": "Property",
        "value": 99,
        "observedAt": "2017-07-29T12:05:02Z",
        "reliability": {
          "type": "Property",
          "value": 0.8
        },
        "providedBy": {
          "type": "Relationship",
          "object": "urn:ngsi-ld:Camera:C2"
        }
      },
      "totalSpotNumber": {
        "type": "Property",
        "value": 100
      },
      "location": {
        "type": "GeoProperty",
        "value": {
          "type": "Point",
          "coordinates": [-8.51, 41.1]
        }
      }
    }
  }
],
```

```
  "@context": [
    "http://example.org/ngsi-ld/latest/parking.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

Concise GeoJSON Representation

The concise GeoJSON representation of a single Entity is defined as a single GeoJSON Feature object as follows:

```
{
  "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      -8.51,
      41.1
    ]
  },
  "properties": {
    "type": "OffStreetParking",
    "name": "Downtown One",
    "availableSpotNumber": {
      "value": 121,
      "observedAt": "2017-07-29T12:05:02Z",
      "reliability": 0.7,
      "providedBy": {
        "object": "urn:ngsi-ld:Camera:C1"
      }
    },
    "location": {
      "type": "Point",
      "coordinates": [
        -8.51,
        41.1
      ]
    },
    "totalSpotNumber": 200,
    "@context": [
      "http://example.org/ngsi-ld/latest/parking.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
}
```

The concise GeoJSON representation of multiple Entities is defined as a GeoJSON FeatureCollection object containing an array of GeoJSON features corresponding to the individual Entity representations in concise GeoJSON format.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -8.5,
          41.1
        ]
      },
      "properties": {
        "type": "OffStreetParking",
        "name": "Downtown One",
        "availableSpotNumber": {
          "value": 121,
          "observedAt": "2017-07-29T12:05:02Z",
          "reliability": 0.7,
          "providedBy": {
            "object": "urn:ngsi-ld:Camera:C1"
          }
        },
        "totalSpotNumber": 200,
        "location": {
          "type": "Point",
          "coordinates": [
```

```
            -8.51,
            41.1
          ]
        }
      }
    },
    {
      "id": "urn:ngsi-ld:OffStreetParking:Downtown2",
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -8.51,
          41.1
        ]
      },
      "properties": {
        "type": "OffStreetParking",
        "name": "Downtown Two",
        "availableSpotNumber": {
          "value": 99,
          "observedAt": "2017-07-29T12:05:02Z",
          "reliability": 0.8,
          "providedBy": {
            "object": "urn:ngsi-ld:Camera:C2"
          }
        },
        "totalSpotNumber": 100,
        "location": {
          "type": "Point",
          "coordinates": [
            -8.51,
            41.1
          ]
        }
      }
    }
  ],
  "@context": [
    "http://example.org/ngsi-ld/latest/parking.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

Simplified GeoJSON Representation

The simplified GeoJSON representation of a single Entity is defined as a single GeoJSON Feature object where the properties represent a collapsed representation of the Entity, which focuses on Property Values and Relationship objects present at the first level of the graph.

```
{
  "id": "urn:ngsi-ld:offstreetparking:Downtown1",
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [-8.51, 41.1]
  },
  "properties": {
    "type": "OffStreetParking",
    "name": "Downtown One",
    "availableSpotNumber": 121,
    "totalSpotNumber": 200,
    "location": {
      "type": "Point",
      "coordinates": [-8.51, 41.1]
    }
  },
  "@context": [
    "http://example.org/ngsi-ld/latest/parking.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

The simplified GeoJSON representation of multiple Entities is defined as a GeoJSON FeatureCollection object containing an array of GeoJSON features corresponding to the individual Entity representations in simplified GeoJSON format.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [-8.5, 41.2]
      },
      "properties": {
        "type": "OffStreetParking",
        "name": "Downtown One",
        "availableSpotNumber": 121,
        "totalSpotNumber": 200,
        "location": {
           "type": "Point",
           "coordinates": [-8.5, 41.2]
        }
      }
    },
    {
      "id": "urn:ngsi-ld:OffStreetParking:Downtown2",
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [-8.51, 41.1]
      },
      "properties": {
        "type": "OffStreetParking",
        "name": "Downtown Two",
        "availableSpotNumber": 99,
        "totalSpotNumber": 100,
        "location": {
           "type": "Point",
           "coordinates": [-8.51, 41.1]
        }
      }
    }
  ],
  "@context": [
    "http://example.org/ngsi-ld/latest/parking.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```


C.2.4 @context

The disposition of the @context can be as an inline JSON object, as a dereferenceable URI or as a (multiple) combination of both. In the examples above the @context is provided through several dereferenceable URIs. The resulting @context (obtained by merging the content of the resource referenced by the referred URIs) is shown below.

NOTE 1: For brevity reasons the @context does not contain the API terms defined by clause 5.2.

NOTE 2: Some extra terms are defined because they will be used in examples later presented.


```
{
    "id": "@id",
    "type": "@type",
    "Property": "https://uri.etsi.org/ngsi-ld/Property",
    "Relationship": "https://uri.etsi.org/ngsi-ld/Relationship",
    "value": "https://uri.etsi.org/ngsi-ld/hasValue",
    "object": {
        "@type": "@id",
        "@id": "https://uri.etsi.org/ngsi-ld/hasObject"
    },
    "observedAt": {
        "@type": "https://uri.etsi.org/ngsi-ld/DateTime",
        "@id": "https://uri.etsi.org/ngsi-ld/observedAt"
    },
    "datasetId": {
        "@id": "https://uri.etsi.org/ngsi-ld/datasetId",
        "@type":"@id"
    },
    "location": "https://uri.etsi.org/ngsi-ld/location",
```

```
    "GeoProperty": "https://uri.etsi.org/ngsi-ld/GeoProperty",
    "Vehicle": "http://example.org/vehicle/Vehicle",
    "street": "http://example.org/vehicle/street",
    "brandName": "http://example.org/vehicle/brandName",
    "category": "http://example.org/vehicle/category",
    "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",
    "commercial": "http://example.org/vehicle/commercial",
    "non-commercial": "http://example.org/vehicle/non-commercial"
}
```

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"
            }
        ],
        "propertyNames": ["brandName","speed"],
        "relationshipNames": ["isParked"]
        },
        {
        "entities": [
            {
                "idPattern": ".*downtown$",
                "type": "OffStreetParking"
            },
            {
                "idPattern": ".*47$",
                "type": "OffStreetParking"
             }
        ],
        "propertyNames": ["availableSpotNumber","totalSpotNumber"],
        "relationshipNames": ["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]] ]
     },
     "managementInterval": {
        "startAt": " 2017-11-29T14:53:15Z"
     },
    "@context": [
        "http://example.org/ngsi-ld/latest/commonTerms.jsonld",
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
```

```
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

The Registration is referring to a Temporal Context Source capable of providing temporal information from Entities of type *Vehicle* and *OffStreetParking*, meeting certain id requirements. More concretely, it can only provide the referenced Properties and Relationships. Temporal information is provided for the given managementInterval, i.e. related to createdAt and modifiedAt Temporal Properties. The managementInterval is specified as an open interval, so only a starting point is given. In addition, the Registration example covers a particular geographical area.

C.4 Context Subscription

Below there is an example of a Context Subscription. It makes use of the @context formerly described.

```
{
    "id": "urn:ngsi-ld:Subscription:mySubscription",
    "type": "Subscription",
    "entities": [
        {
            "type": "Vehicle"
        }
    ],
    "watchedAttributes": ["speed"],
    "q": "speed>50",
    "geoQ": {
        "georel": "near;maxDistance==2000",
        "geometry": "Point",
        "coordinates": [-1,100]
    },
    "notification": {
        "attributes": ["speed"],
        "format": "keyValues",
        "endpoint": {
            "uri": "http://my.endpoint.org/notify",
            "accept": "application/json"
        }
    },
    "@context": [
        "http://example.org/ngsi-ld/latest/vehicle.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

The subject of the Context Subscription is 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. ETSI ISG CIM published a Developer's Primer with many more examples, see ETSI GR CIM 008 [i.21].

C.5.2 Create Entity of Type Vehicle

C.5.2.1 HTTP Request

POST /ngsi-ld/v1/entities/

Content-Type: application/ld+json

Content-Length: 556

<Insert Here the JSON-LD representation of a Vehicle as described by clause C.2.2 Vehicle Entity>

C.5.2.2 HTTP Response

201 Created

Location: /ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:A4567

C.5.3 Query Entities

C.5.3.1 Introduction

EXAMPLE: Give back all the Entities of type *Vehicle* whose "brandName" attribute is not "Mercedes". Only give back the "brandName" attribute and provide the data in the NGSI-LD Simplified Format.

C.5.3.2 HTTP Request

GET /ngsi-ld/v1/entities/?type=Vehicle&q=brandName!="Mercedes"&options=keyValues

Accept: application/ld+json

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

C.5.3.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "brandName":"Volvo",
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```

C.5.4 Query Entities (Pagination)

C.5.4.1 Introduction

EXAMPLE: Give back all the Entities of type *Vehicle*. Only give back the "brandName" attribute and provide the data in the NGSI-LD Simplified Format. Limit the number of entities retrieved to 2.

C.5.4.2 HTTP Request

GET /ngsi-ld/v1/entities/?type=Vehicle&options=keyValues&limit=2

Accept: application/ld+json

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

C.5.4.3 HTTP Response

200 OK

Content-Type: application/ld+json

Link: </ngsi-ld/v1/entities/?type=Vehicle&options=keyValues&limit=2&offset=2>; rel="next"; type="application/ld+json"

```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "brandName":"Volvo",
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  },
  {
    "id":"urn:ngsi-ld:Vehicle:A456",
    "type":"Vehicle",
    "brandName":"Mercedes",
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```

C.5.5 Temporal Query

C.5.5.1 Introduction

EXAMPLE 1: Give back the temporal evolution of the attribute "speed" of Entities of type *Vehicle* whose "brandName" attribute is not "Mercedes" between the 1st of August at noon and the 1st of August at 01 PM.

EXAMPLE 2: Give back the temporal evolution of the attribute "speed" and "brandName" of Entities of type *Vehicle* whose "brandName" attribute is not "Mercedes" between the 1st of August at noon and the 1st of August at 01 PM. As "brandName" attribute does not have any temporal evolution, "brandName" attribute is omitted in the response.

C.5.5.2 HTTP Request #1

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&q=brandName!=Mercedes&attrs=speed&timerel=between&timeAt=2018-08-01T12:00:00Z&endTimeAt=2018-08-01T13:00:00Z

Accept: application/ld+json

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

C.5.5.3 HTTP Response #1

200 OK

Content-Type: application/ld+json

```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "speed":[
      {
```

```
        "type":"Property",
        "value":120,
        "observedAt":"2018-08-01T12:03:00Z"
      },
      {
        "type":"Property",
        "value":80,
        "observedAt":"2018-08-01T12:05:00Z"
      },
      {
        "type":"Property",
        "value":100,
        "observedAt":"2018-08-01T12:07:00Z"
      }
    ],
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```


C.5.5.2 HTTP Request #2

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&q=brandName!=Mercedes&attrs=speed,brandName&timerel=between&timeAt=2018-08-01T12:00:00Z&endTimeAt=2018-08-01T13:00:00Z

Accept: application/ld+json

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

C.5.5.3 HTTP Response #2

200 OK

Content-Type: application/ld+json


```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "speed":[
      {
        "type":"Property",
        "value":120,
        "observedAt":"2018-08-01T12:03:00Z"
      },
      {
        "type":"Property",
        "value":80,
        "observedAt":"2018-08-01T12:05:00Z"
      },
      {
        "type":"Property",
        "value":100,
        "observedAt":"2018-08-01T12:07:00Z"
      }
    ],
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```

C.5.6 Temporal Query (Simplified Representation)

C.5.6.1 Introduction

EXAMPLE: Give back the temporal evolution of the "speed" attribute for Entities of type *Vehicle* whose "brandName" attribute is not "Mercedes" between the 1st of August at noon and the 1st of August at 01 PM. Simplified representation is required.

C.5.6.2 HTTP Request

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&q=brandName!=Mercedes&attrs=speed&timerel=between&timeAt=2018-08-01T12:00:00Z&endTimeAt=2018-08-01T13:00:00Z&options=**temporalValues**

Accept: application/ld+json

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

C.5.6.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "speed":{
      "type":"Property",
      "values":[
        [
          120,
          "2018-08-01T12:03:00Z"
        ],
        [
          80,
          "2018-08-01T12:05:00Z"
        ],
        [
          100,
          "2018-08-01T12:07:00Z"
        ]
      ]
    },
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```

C.5.7 Retrieve Available Entity Types

C.5.7.1 Introduction

EXAMPLE: Give back all entity types for which entity instances are currently available in the NGSI-LD system.

C.5.7.2 HTTP Request

GET /ngsi-ld/v1/types

Accept: application/json

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

C.5.7.3 HTTP Response

200 OK

Content-Type: application/json

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

```
{
  "id":"urn:ngsi-ld:EntityTypeList:34534657",
  "type":"EntityTypeList",
  "typeList":[
    "Vehicle",
    "OffStreetParking",
    "http://example.org/parking/ParkingSpot"
  ]
}
```

NOTE: All entity types that can be found in the provided @context are given as short names, the others as Fully Qualified Names (FQNs).

C.5.8 Retrieve Details of Available Entity Types

C.5.8.1 Introduction

EXAMPLE: Give back the details of all entity types for which entity instances are currently available in the NGSI-LD system.

C.5.8.2 HTTP Request

GET /ngsi-ld/v1/types?details=true

Accept: application/json

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

C.5.8.3 HTTP Response

200 OK

Content-Type: application/json

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

```
[
  {
    "id":"http://example.org/vehicle/Vehicle",
    "type":"EntityType",
    "typeName":"Vehicle",
    "attributeNames":[
      "brandName",
      "isParked",
      "location",
```

```
      "speed"
    ]
  },
  {
    "id":"http://example.org/parking/OffStreetParking",
    "type":"EntityType",
    "typeName":"OffStreetParking",
    "attributeNames":[
      "availableSpotNumber",
      "isNextToBuilding",
      "location",
      "totalSpotNumber"
    ]
  },
  {
    "id":"http://example.org/parking/ParkingSpot",
    "type":"EntityType",
    "typeName":"http://example.org/parking/ParkingSpot",
    "attributeNames":[
      "location",
      "http://example.org/parking/status"
    ]
  }
]
```

NOTE: The type name of all entity types and all attribute names that can be found in the provided @context are given as short names, the others as Fully Qualified Names (FQNs). The id is always an FQN.

C.5.9 Retrieve Available Entity Type Information

C.5.9.1 Introduction

EXAMPLE: Give back the details of entity type *Vehicle* (for which entity instances are currently available in the NGSI-LD system).

C.5.9.2 HTTP Request

GET /ngsi-ld/v1/types/Vehicle

[Alternative with FQN: **GET** /ngsi-ld/v1/attributes/http%3A%2F%2Fexample.org%2Fvehicle%2FVehicle]

Accept: application/json

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

C.5.9.3 HTTP Response

200 OK

Content-Type: application/json

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

```
{
  "id":"http://example.org/vehicle/Vehicle",
  "type":"EntityTypeInfo",
  "typeName":"Vehicle",
  "entityCount":2,
  "attributeDetails":[
    {
      "id":"http://example.org/vehicle/brandName",
      "type":"Attribute",
      "attributeName":"brandName",
      "attributeTypes":[
        "Property"
      ]
    },
```

```
    {
      "id":"http://example.org/vehicle/isParked",
      "type":"Attribute",
      "attributeName":"isParked",
      "attributeTypes":[
        "Relationship"
      ]
    },
    {
      "id":"https://uri.etsi.org/ngsi-ld/location",
      "type":"Attribute",
      "attributeName":"location",
      "attributeTypes":[
        "GeoProperty"
      ]
    },
    {
      "id":"http://example.org/vehicle/speed",
      "type":"Attribute",
      "attributeName":"speed",
      "attributeTypes":[
        "Property"
      ]
    }
  ]
}
```

C.5.10 Retrieve Available Attributes

C.5.10.1 Introduction

EXAMPLE: Give back all attribute names for which entity instances are currently available in the NGSI-LD system that have an attribute with the respective name.

C.5.10.2 HTTP Request

GET /ngsi-ld/v1/attributes

Accept: application/json

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

C.5.10.3 HTTP Response

200 OK

Content-Type: application/json

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

```
{
  "id":"urn:ngsi-ld:AttributeList:56534657",
  "type":"AttributeList",
  "attributeList":[
    "brandName",
    "isParked",
    "location",
    "speed",
    "http://example.org/parking/status"
  ]
}
```

NOTE: The attribute names that can be found in the provided @context are given as short names, the others as Fully Qualified Names (FQNs).

C.5.11 Retrieve Details of Available Attributes

C.5.11.1 Introduction

EXAMPLE: Give back the details of all attributes for which entity instances are currently available in the NGSI-LD system to which an attribute with the respective attribute name belongs.

C.5.11.2 HTTP Request

GET /ngsi-ld/v1/attributes?details=true

Accept: application/json

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

C.5.11.3 HTTP Response

200 OK

Content-Type: application/json

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

```
[
  {
    "id":"http://example.org/vehicle/brandName",
    "type":"Attribute",
    "attributeName":"brandName",
    "typeNames":[
      "Vehicle"
    ]
  },
  {
    "id":"http://example.org/vehicle/isParked",
    "type":"Attribute",
    "attributeName":"isParked",
    "typeNames":[
      "Vehicle"
    ]
  },
  {
    "id":"https://uri.etsi.org/ngsi-ld/location",
    "type":"Attribute",
    "attributeName":"location",
    "typeNames":[
      "Vehicle",
      "OffStreetParking",
      "http://example.org/parking/ParkingSpot"
    ]
  },
  {
    "id":"http://example.org/vehicle/speed",
    "type":"Attribute",
    "attributeName":"speed",
    "typeNames":[
      "Vehicle"
    ]
  },
  {
    "id":"http://example.org/parking/status",
    "type":"Attribute",
    "attributeName":"http://example.org/parking/status",
    "typeNames":[
      "http://example.org/parking/ParkingSpot"
    ]
  }
]
```

NOTE: The attribute name and all type names that can be found in the provided @context are given as short names, the others as Fully Qualified Names (FQNs). The id is always an FQN.

C.5.12 Retrieve Available Attribute Information

C.5.12.1 Introduction

EXAMPLE: Give back the details of the attribute named "brandName" (for which entity instances with an attribute of this name are currently available in the NGSI-LD system).

C.5.12.2 HTTP Request

GET /ngsi-ld/v1/attributes/brandName

[Alternative with FQN: **GET** /ngsi-ld/v1/attributes/http%3A%2F%2Fexample.org%2Fvehicle%2FbrandName]

Accept: application/json

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

C.5.12.3 HTTP Response

200 OK

Content-Type: application/json

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

```
{
  "id": "http://example.org/vehicle/brandName",
  "type": "Attribute",
  "attributeName": "brandName",
  "attributeTypes": ["Property"],
  "typeNames": ["Vehicle"],
  "attributeCount": 2
}
```

C.5.13 Query Entities (Natural Language Filtering)

C.5.13.1 Introduction

EXAMPLE: Give back all the Entities of type *Vehicle* where the "marque" attribute in British English is "Vauxhall Viva". Only give back the "marque" attribute and provide the data in the NGSI-LD Simplified Format and only return language strings in German.

C.5.13.2 HTTP Request

GET /ngsi-ld/v1/entities/?type=Vehicle&attrs=marque&q=marque[en-GB]=="Vauxhall Viva"&options=keyValues&lang=de

Accept: application/ld+json

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

C.5.13.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
  {
    "id":"urn:ngsi-ld:Vehicle:A4567",
    "type":"Vehicle",
    "marque":"Opel Karl",
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```

C.5.14 Temporal Query (Aggregated Representation)

C.5.14.1 Introduction

EXAMPLE: Give back the maximum and average speed of Entities of type *Vehicle* whose "brandName" attribute is not "Mercedes" between the 1st of August at noon and the 1st of August at 01 PM, aggregated by periods of 4 minutes.

C.5.14.2 HTTP Request

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&q=brandName!=Mercedes&attrs=speed&timerel=between&timeAt=2018-08-01T12:00:00Z&endTimeAt=2018-08-01T13:00:00Z&aggrMethods=max,avg&aggrPeriodDuration=PT4M&options=**aggregatedValues**

Accept: application/ld+json

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

C.5.14.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "speed":{
      "type":"Property",
      "max":[
        [
          120,
          "2018-08-01T12:00:00Z",
          "2018-08-01T12:04:00Z"
        ],
        [
          100,
          "2018-08-01T12:04:00Z",
          "2018-08-01T12:08:00Z"
        ]
      ],
      "avg":[
        [
          120,
          "2018-08-01T12:00:00Z",
          "2018-08-01T12:04:00Z"
        ],
```

```
        [
          90,
          "2018-08-01T12:04:00Z",
          "2018-08-01T12:08:00Z"
        ]
      ]
    },
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  }
]
```

C.5.15 Scope Queries

C.5.15.1 Introduction

EXAMPLE: Give back all the Entities of type *OffStreetParking* that are within the Scope /Madrid/Centro or /Madrid/Cortes.

C.5.15.2 HTTP Request

GET /ngsi-ld/v1/entities/?type=OffStreetParking&scopeQ="/Madrid/Centro,/Madrid/Cortes"

Accept: application/ld+json

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

C.5.15.3 HTTP Response

200 OK

Content-Type: application/ld+json

```
[
   {
      "id":"urn:ngsi-ld:OffStreetParking:Downtown1",
      "type":"OffStreetParking",
      "scope":"/Madrid/Centro",
      "name":{
         "type":"Property",
         "value":"Downtown One"
      },
      "availableSpotNumber":{
         "type":"Property",
         "value":121,
         "observedAt":"2017-07-29T12:05:02Z",
         "reliability":{
            "type":"Property",
            "value":0.7
         },
         "providedBy":{
            "type":"Relationship",
            "object":"urn:ngsi-ld:Camera:C1"
         }
      },
      "totalSpotNumber":{
         "type":"Property",
         "value":200
      },
      "location":{
         "type":"GeoProperty",
         "value":{
            "type":"Point",
            "coordinates":[
               -8.5,
               41.2
            ]
```

```
            }
         },
         "@context":[
            "http://example.org/ngsi-ld/latest/parking.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
         ]
      },
      {
         "id":"urn:ngsi-ld:OffStreetParking:Corte4",
         "type":"OffStreetParking",
         "scope":[
               "/Madrid/Cortes",
               "/Company894/UnitC"
         ],
         "name":{
            "type":"Property",
            "value":"Corte4"
         },
         "availableSpotNumber":{
            "type":"Property",
            "value":121,
            "observedAt":"2017-07-29T12:05:02Z",
            "reliability":{
               "type":"Property",
               "value":0.7
            },
            "providedBy":{
               "type":"Relationship",
               "object":"urn:ngsi-ld:Camera:C1"
            }
         },
         "totalSpotNumber":{
            "type":"Property",
            "value":100
         },
         "location":{
            "type":"GeoProperty",
            "value":{
               "type":"Point",
               "coordinates":[
                  -8.6,
                  41.3
               ]
            }
         },
         "@context":[
            "http://example.org/ngsi-ld/latest/parking.jsonld",
            "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
         ]
      }
]
```


C.5.16 Temporal Scope Queries

C.5.16.1 Introduction

EXAMPLE: Give back the speed of all the Entities of type *Vehicle* that have been within the Scope /Madrid/Centro between the 1st of August 2018 at noon and the 1st of August 2018 at 01 PM. Note that the value of the Scope has to match for the given timeframe, which means it is possible that it has been set before, e.g. on 1st of August 2018 at 11 AM.

C.5.16.2 HTTP Request

GET /ngsi-ld/v1/temporal/entities/?type=Vehicle&attrs=speed,scope&timerel=between&timeAt=2018-08-01T12:00:00Z&endTimeAt=2018-08-01T13:00:00Z&scopeQ="/Madrid/Centro"

Accept: application/ld+json

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

C.5.16.3 HTTP Response

200 OK

Content-Type: application/ld+json


```
[
  {
    "id":"urn:ngsi-ld:Vehicle:B9211",
    "type":"Vehicle",
    "scope":{
      "type":"Property",
      "values":[
        [
          "/Madrid/Centro",
          "2018-08-01T11:00:00Z"
        ]
      ]
    },
    "speed":{
      "type":"Property",
      "values":[
        [
          30,
          "2018-08-01T12:03:00Z"
        ],
        [
          60,
          "2018-08-01T12:05:00Z"
        ],
        [
          50,
          "2018-08-01T12:07:00Z"
        ]
      ]
    },
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
  },
  {
    "id":"urn:ngsi-ld:Vehicle:A8311",
    "type":"Vehicle",
    "scope":{
      "type":"Property",
      "values":[
        [
          [
             "/Madrid/Centro",
             "/Company123/UnitA"
          ],
          "2018-08-01T12:10:00Z"
        ]
      ]
    },
    "speed":{
      "type":"Property",
      "values":[
        [
          40,
          "2018-08-01T12:12:00Z"
        ],
        [
          60,
          "2018-08-01T12:14:00Z"
        ],
        [
          50,
          "2018-08-01T12:16:00Z"
        ]
      ]
    },
    "@context":[
      "http://example.org/ngsi-ld/latest/vehicle.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
```

```
    }
  ]
```

Vehicle B9211 has already been within the Scope /Madrid/Centro before the beginning of the request interval, whereas Vehicle A8311 only entered the Scope within the request interval. Thus in the latter case only Property values are included that have been observed after the Scope has become valid.

C.6 Date Representation

In NGSI-LD, a *TemporalProperty* is represented only by its value, i.e. no sub-Properties of *TemporalProperty* nor sub-Relationships of *TemporalProperty* can be conveyed. In more formal language, a *TemporalProperty* does not allow reification. The term *TemporalProperty* has been reserved for non-reified structural timestamps (*observedAt*, *createdAt*, *modifiedAt*, *deletedAt*), which capture the temporal evolution of Attributes. Only such structural timestamps can be used as *timeproperty* in Temporal Queries as mandated by clause 4.11.

The following example shows how time values (*Date*, *Time*, or *DateTime*) can be represented in NGSI-LD as reified Properties. A reified Property whose value is assigned the JSON type *Date, DateTime or Time*, can use the JSON-LD *@value* syntax structure, as shown by the example below:

```
{
  "id": "urn:ngsi-ld:Vehicle:B9211",
  "type": "Vehicle",
  "testedAt": {
    "type": "Property",
    "value": {
      "@type": "DateTime",
      "@value": "2018-12-04T12:00:00Z"
    }
  },
  "@context": [
    "http://example.org/ngsi-ld/latest/vehicle.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

One other way to achieve the same result would be to use JSON-LD "type coercion". With type coercion, values with a special data type are defined with *@type* in the *@context*. This enforces the correct type for any occurrence. An example of such a @context fragment is shown below:

```
"testedAt": {
  "@type": "https://uri.etsi.org/ngsi-ld/DateTime",
  "@id": "http://example.org/test/testedAt"
}
```

The above does not work, when using the @context to perform compaction, in the normalized and compact representation of NGSI-LD, due to reification of the Property, because in this case *testedAt* is a complex JSON object, which cannot be compacted to a DateTime type as the @context specifies. Thus, the full URI *http://example.org/test/testedAt* is kept, instead of the short name *testedAt*. In summary, user @contexts used for the normalized and compact NGSI-LD representation cannot use the JSON-LD type coercion feature.

However, in the simplified (keyValue) representation case, such an @context with the specification of *testedAt* could be used, as there is no reification.

As a side note, when using the above *@value* + *@type* approach, since *type* is mapped to *@type* in the NGSI-LD core *@context*, JSON-LD compaction will result in the following compacted value, instead of the one shown above, because *@type* is compacted to *type*:

```
"value": {
  "type": "DateTime",
  "@value": "2018-12-04T12:00:00Z"
}
```

C.7 @context utilization clarifications

When expanding or compacting JSON-LD terms, the JSON-LD @context to be used is always the one provided in the current API request. For the benefit of users and implementers the following examples illustrate this concept.

The scenario starts with the creation of an Entity using a JSON-LD @context as follows:

POST /ngsi-ld/v1/entities/

Content-Type: application/ld+json

Content-Length: 200

```
{
  "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
  "type": "OffStreetParking",
  "name": {
    "type": "Property",
    "value": "Downtown One"
  },
  "availableSpotNumber": {
    "type": "Property",
    "value": 121,
    "observedAt": "2017-07-29T12:05:02Z"
  },
  "totalSpotNumber": {
    "type": "Property",
    "value": 200
  },
  "@context": [
    "http://example.org/ngsi-ld/latest/parking.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

The content of the @context utilized for the referred Entity creation (at http://example.org/ngsi-ld/latest/parking.jsonld) is as follows:

```
{
   "OffStreetParking": "http://example.org/parking/OffStreetParking",
   "availableSpotNumber": "http://example.org/parking/availableSpotNumber",
   "totalSpotNumber": "http://example.org/parking/totalSpotNumber",
   "name": "http://example.org/parking/name"
}
```

At Entity creation time the implementation will perform the expansion of terms using the JSON-LD @context depicted above.

Now it is needed to retrieve our initial Entity. For retrieving such Entity, this time, a different JSON-LD @context is going to be utilized, as follows:

```
{
   "OffP": "http://example.org/parking/OffStreetParking",
   "ava": "http://example.org/parking/availableSpotNumber",
   "total": "http://example.org/parking/totalSpotNumber"
}
```

This new @context, even though it makes use of the same set of Fully Qualified Names, is defining new short strings as terms. The reasons for that could be multiple: to facilitate data consumption by clients, to save some bandwidth, to enable a more (or less) human-readable response payload body in a language different than English, etc.

In this particular case, the result of the Entity retrieval will be as depicted below. It can be observed that the terms defined by the JSON-LD @context **provided at retrieval time** are used to render the Entity content (compaction), and **not** the terms that were provided at creation time (which may be no longer known by the NGSI-LD Broker).

It is also interesting to note that the @context array of the response payload body contains, indeed, our header-supplied @context:

GET /ngsi-ld/v1/entities/urn:ngsi-ld:OffStreetParking:Downtown1

Accept: application/ld+json

Link: <http://example.org/ngsi-ld/latest/parking-abbreviated.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

```
{
  "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
  "type": "OffP",
  "name": {
    "type": "Property",
    "value": "Downtown One"
  },
  "ava": {
    "type": "Property",
    "value": 121,
    "observedAt": "2017-07-29T12:05:02Z"
  },
  "total": {
  "type": "Property",
    "value": 200
     },
  "@context": [
    "http://example.org/ngsi-ld/latest/parking-abbreviated.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
  ]
}
```

Another interesting case to note is the one when an @context with no matching terms or no @context at all is supplied. See the following example:

GET /ngsi-ld/v1/entities/urn:ngsi-ld:OffStreetParking:Downtown1

Accept: application/ld+json

```
{
  "id": "urn:ngsi-ld:OffStreetParking:Downtown1",
  "type": "http://example.org/parking/OffStreetParking",
  "http://example.org/parking/name": {
    "type": "Property",
    "value": "Downtown One"
  },
  "http://example.org/parking/availableSpotNumber": {
    "type": "Property",
    "value": 121,
    "observedAt": "2017-07-29T12:05:02Z"
  },
  "http://example.org/parking/totalSpotNumber": {
    "type": "Property",
    "value": 200
  },
  "@context": "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
}
```

In this particular case it can be observed that the user names (Entity Type, Attributes) in the response payload body have not been compacted, and as a result the Fully Qualified Names are included. However, the core API terms have been compacted, as the Core @context is always considered to be implicitly present if not specified explicitly (and that is why it is included in the JSON-LD response, as mandated by the specification).

C.8 Link header utilization clarifications

The JSON-LD Specification [2] states clearly that **only one HTTP Link header** with the link relationship <http://www.w3.org/ns/json-ld#context> is required to appear. Such statement has implications in terms of providing the JSON-LD @context when using the NGSI-LD API. The main implication is that if the @context is a compound one, i.e. an @context which references multiple individual @context, served by resources behind different URIs, then a **wrapper** @context has to be created and hosted. The final aim is that only one @context is referenced from the JSON-LD Link header. This can be illustrated with an example:

Imagine that it is desired to create an Entity providing @context terms which are defined in two different JSON-LD @context resources:

- http://example.org/vehicle/v1/vehicle-context.jsonld

- https://schema.org

If a developer wants to reference these two @context resources from a Link header, a wrapper @context can be easily created as follows:

```
{
   "@context": [
       "http://example.org/vehicle/v1/vehicle-context.jsonld",
       "https://schema.org",
       "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
   ]
}
```

As such wrapper @context needs to be referenced from a Link header by using a URI, then it will have to be hosted at some place on the Web. Usually, developers will host @context using popular and simple solutions such as Github or Gitlab pages. As a result, developers will be able to use @context in queries or when using "application/json" as main content type managed by their applications.

It is a **good practice to include the Core @context** in the wrapper @context so it can be used, off-the-shelf, by external JSON-LD processing tools. However, it should be noted this is not necessary for NGSI-LD, as the Core @context is always implicitly included.

Then, using such wrapper @context, (in our example hosted at *https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld*), the developer will be able to issue requests like:

POST /ngsi-ld/v1/entities/

Content-Type: application/json

Content-Length: 200

Link: <https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

```
{
    "id": "urn:ngsi-ld:Vehicle:V1",
    "type": "Vehicle",
    "builtYear": {
        "type": "Property",
        "value": "2014"
    },
    "speed": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    }
}
```

201 Created

Location: /ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:V1

Link: < https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld >; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

GET /ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:V1

Accept: application/ld+json

Link: <https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld>; rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"

200 OK

Content-Type: application/ld+json

```
{
    "id": "urn:ngsi-ld:Vehicle:V1",
    "type": "Vehicle",
    "builtYear": {
        "type": "Property",
```

```
        "value": "2014"
    },
    "speed": {
        "type": "Property",
        "value": 121,
        "observedAt": "2017-07-29T12:05:02Z"
    },
    "@context": "https://hosting.example.com/ngsi-ld/v1/wrapper-context.jsonld"
}
```

Observe that in this case the NGSI-LD Broker is responding with the same wrapper @context in the Link header of the HTTP Response or within the JSON-LD response payload body (when MIME type accepted is "application/ld+json"). However, that could not be always the case, as there could be situations where the NGSI-LD Broker could need to provide a wrapper @context hosted by itself, for instance, when there are inline @context terms or when the Core @context has not been previously included by the wrapper @context (not recommended) provided within developer's requests.

C.9 @context processing clarifications

JSON-LD Specification [2] says that "If a term is redefined within a context, all previous rules associated with the previous definition are removed". In addition, it is stated that "Multiple contexts may be combined using an array, which is processed in order".

In contrast to the JSON-LD Specification, the NGSI-LD specification states that the Core @context is protected and has to remain immutable. This essentially means that the Core @context has final precedence and, therefore, is always to be processed as the last one in the @context array. For clarity, data providers should place the Core @context in the final position. From the point of view of Data providers, care has to be taken so that there are no unexpected or undesired term expansions. See the following example:

```
{
    "id": "urn:ngsi-ld:Building:B1",
    "type": "Building",
    "name": {
        "type": "Property",
        "value": "Empire State"
    },
    "openingHours": {
        "type": "Property",
        "value": "Mo-Fr 10am-7pm Sa 10am-22pm Su 10am-21pm"
    },
    "@context": [
        "https://schema.org",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

The main caveat of the example above is that the term "name" is defined in multiple elements of the @context and the last one takes final precedence for the expansion. In these situations, one solution is to prefix the conflicting terms, so that there cannot be any clashing. Therefore, if the intent is to refer to https://schema.org/name throughout, the example above can be modified as shown below:

```
{
    "id": "urn:ngsi-ld:Building:B1",
    "type": "Building",
    "schema:name": {
        "type": "Property",
        "value": "Empire State"
    },
    "openingHours": {
        "type": "Property",
        "value": "Mo-Fr 10am-7pm Sa 10am-22pm Su 10am-21pm"
    },
    "@context": [
        "https://schema.org",
        "http://example.org/ngsi-ld/latest/parking.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
    ]
}
```

Note that the Core @context should be placed in the last position of the @context array. NGSI-LD implementations are required to render content following this approach, which has been undertaken in order to maximize compatibility with JSON-LD processing tools. This example works because the "schema:" prefix has already been defined by the schema.org @context.

Annex D (informative): Transformation Algorithms

D.1 Introduction

These algorithms are informative but NGSI-LD implementations should aim at either implementing them as they are described here or devising similar algorithms which take exactly the same input and provides exactly the same output (or an equivalent one as per the JSON-LD specification [2]).

D.2 Algorithm for transforming an NGSI-LD Entity into a JSON-LD document (ALG1)

This algorithm takes as input an NGSI-LD graph which top level node is a particular Entity and returns as output a JSON-LD document which represents all the data associated to the entity. The JSON-LD document (and its associated @context) corresponds to a representation of the Entity in JSON-LD as per the NGSI-LD Information Model.

NOTE: An early implementation of this algorithm can be found at [i.5].

Let:

- **G** be a graph defined as follows:
  - Let **N** be G's top level node.
  - N is an Entity instance of type **T** or types **Ts**. Type Name is "AliasT" or there is an Array of Type Names ["AliasT1", …,"AliasTn"], 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 name 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> or new members <"AliasT1", T1> … <"AliasTn", Tn>.
2) Add to O two new members:
   a) <"id", I>.
   b) <"type", "AliasT"> or <"type", ["AliasT1", …,"AliasTn"]> .">.

3) For each Property Psi (Pi, "AliasP", Vi, Di) associated to N:

   a) Run Algorithm *ALG1.1* taking the following inputs:

      - Ps → Psi.
      - O → O.
      - C → C.

4) For each Relationship Rs (Ri, AliasRi, Robji) associated to N:

   a) Run Algorithm *ALG1.2* taking the following inputs:

      - Rs → Rsi.
      - O → O.
      - C → C.

5) Return (O, C) and end of the algorithm.

D.3 Algorithm for transforming an NGSI-LD Property into JSON-LD (ALG1.1)

Let **Ps** be the Property that has to be transformed. It is defined by (P, "AliasP", V, D), where **P** denotes a Property Type Id, "**AliasP**" is the Property Name, **V** is the Property Value and **D** is the Property Value's data type.

Ps might be associated to extra Properties or Relationships.

Let O be the output JSON-LD object and C the associated JSON-LD context:

1) Execute the following steps:

   a) If no member with "AliasP" is present in O, add a new member to O with key "AliasP" and value an object structure, let it be named **Op** as defined in the following. Otherwise, add all existing members with "AliasP" to a JSON-LD array and in addition put the object structure Op as defined in the following:

      - <"type", "Property">.
      - If D is not a native JSON data type add a new member to Op with name "value" and which value has to be an object structure as follows:

        1) <"@type", D>.
        2) <"@value", V>.

      - Else If D is a native JSON data type add a new member to Op as follows:

        1) <"value", V>.

   b) Add a new member to C as follows:

      - <"AliasP", P>.

   c) For each Property associated to Ps (Pss) recursively run the present algorithm (*ALG1.1*) taking the following inputs:

      - Ps → Pss.
      - O → Op.
      - C → C.

d) For each Relationship associated to Ps (Rss) run algorithm *ALG1.2* taking the following inputs:

- Rs → Rss.
- O → Op.
- C → C.

2) Return (O,C) and end of the algorithm.

D.4 Algorithm for transforming an NGSI-LD Relationship into JSON-LD (ALG1.2)

Let **Rs** be the Relationship that has to be transformed. It is defined by (R, "AliasR", Robj), where **R** denotes a Relationship Type Id, "**AliasR**" is the Relationship's Name and **Robj** is the identifier of the target object of the Relationship.

Rs might be associated to extra Properties or Relationships.

Let O be the output JSON-LD object and C the current JSON-LD context:

1) Execute the following statements:

a) If no member with "AliasR" is present in O, add a new member to O with key "AliasR" and value an object structure, let it be named **Or**, and defined as in the following. Otherwise, add all existing members with "AliasR" to a JSON-LD array and in addition put the object structure Or as defined in the following:

- <"object", Robj>.
- <"type", "Relationship">.

b) For each Property associated to Rs (Pss) run the algorithm *ALG1.1* taking the following inputs:

- Ps → Pss.
- O → Or.
- C → C.

c) For each Relationship associated to Rs (Rss) recursively run the present algorithm *ALG1.2* taking the following inputs:

- Rs → Rss.
- O → Or.
- C → C.

2) Return (O,C) and end of the algorithm.

Annex E (informative): RDF-compatible specification of NGSI-LD meta-model

The content of this annex is now in ETSI GS CIM 006 [i.8].

Annex F (informative): Conventions and syntax guidelines

When new concepts or terms are defined they are marked in bold.

EXAMPLE 1: **NGSI-LD Entity, Query Term, observedAt**.

API Parameter names are always in lowercase.

EXAMPLE 2: Options.

Entity Types, JSON-LD node types and Data Types are defined using lowercase but with a starting capital letter.

EXAMPLE 3: Vehicle, Property, Relationship, DateTime.

JSON-LD terms are always defined using camel case notation starting with lower case.

EXAMPLE 4: createdAt, value, unitCode.

When referring to special terms or words, defined previously in the present document or by other referenced specifications, italics format is used.

EXAMPLE 5: *GeoProperty*, *Geometry*, *Second*, *Number*.

When referring to literal strings double quotes are used.

EXAMPLE 6: "application/json", "Subscription".

When referring to the JSON-LD Context the mnemonic text string @context is used as a placeholder.

All the dates and times are given in UTC format.

EXAMPLE 7: 2018-02-09T11:00:00Z.

The measurement units used in the API are those defined by the International System of Units.

EXAMPLE 8: The distance in geo-queries is provided in meters.

When defining application-specific elements or API extensions the same conventions and syntax guidelines should be followed.

Annex G (informative): Localization and Internationalization Support

G.0 Foreword

These algorithms described below are informative, but NGSI-LD implementations should aim at either implementing them as they are described here or providing equivalent @context elements for their payloads to provide interoperability with their internationalized context entities.

G.1 Introduction

G.1.0 Foreword

Since Internationalization is not core to context information management, any direct support within NGSI-LD systems is limited. Annex G proposes a series of best practices for maintaining, querying and displaying interoperable internationalized data.

The content of the @context utilized for the referred Entities within these examples uses pre-existing URNs used for internationalization and is as follows:

```
{
   "inLanguage": "http://schema.org/inLanguage",
   "sameAs": "http://schema.org/sameAs"
}
```

G.1.1 Associating an Entity with a Natural Language

Where a context Entity is associated with a single natural language, include a well-defined Property indicating the natural language of the content. For example an Event taking place in French may be defined as follows:

```
{
    "type": "Event",
    "id": "urn:ngsi-ld:Event:bonjourLeMonde",
    "name": {
        "type": "Property",
        "value": "Bonjour le Monde"
    },
    "description": {
         "type": "Property",
         "value": "«Bonjour le monde» sont les mots traditionnellement écrits par un programme
informatique simple"
    },
    "inLanguage": {
        "type": "Property",
        "value": "fr"
    }
}
```

G.1.2 Associating a Property with a Natural Language

Where a Property of a context entity can be associated to one more natural language, include additional metadata as a sub-Property of that Property. For example, a Hotel with booking forms available in English, French and German may be defined as follows:

```
{
    "type": "Hotel,
    "id": "urn:ngsi-ld:Hotel:XXXXX",
    "name": {
        "type": "Property",
        "value": "Grand Hotel"
```

```
    },

    "bookingUrl": {
        "type": "Property",
        "value": [
            "http://example.com/booking-in-french/",
            "http://example.com/booking-in-english/",
            "http://example.com/booking-in-german/"
        ],
        "inLanguage": {
            "type": "Property",
            "value": ["fr", "en", "de" ]
        }
    }
}
```

G.1.3 Associating as equivalent entity

Where equivalent context entities in multiple natural languages exist, they may be associated with each other through the use of a one-to-many relationship, where each relationship holds an additional sub-Property indicating the natural language of the equivalent entities.

For example, three Events (such as a walking tour which is available in English, French and German) may be associated to each other as follows:

```
{
    "type": "Event",
    "id": "urn:ngsi-ld:Event:bonjourLeMonde",
    "name": {
        "type": "Property",
        "value": "Bonjour le Monde"
    },
    "sameAs": [
        {
            "type": "Relationship",
            "datasetId" : "urn:ngsi-ld:Relationship:1",
            "object": "urn:ngsi-ld:Event:helloWorld",
            "inLanguage": {
                    "type": "Property",
                    "value": "en"
            }
        },
        {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Event:halloWelt",
            "inLanguage": {
                    "type": "Property",
                    "value": "de"
            }
        }
    ]
 }
```

G.2 Natural Language Collation Support

G.2.0 Foreword

All strings within an NGSI-LD system are defined and sorted as a sequence of Unicode characters. As such there is no simple collation mechanism to query entities ignoring case, diacritic marks or matching diphthong single letters such as the German `"ö"` to also match with `"oe"`.

Many databases support a degree of natural language support, in general collation support will always depend upon the underlying database and as such will vary from implementation to implementation. This therefore and cannot be standardized and exposed as part of the context information management API. Furthermore, collation is slow and processor intensive, and for massive systems is better achieved using a separate index.

For systems that require it, this clause proposes a mechanism as an extension to a NGSI-LD broker which can be modified and used to offer collation support to the natural language attributes found within context entities where necessary through creating, querying and maintaining an additional property of a property for collated attributes.

G.2.1 Maintain collations as metadata

- Create a subscription on the attribute (e.g. `name`)
- Create a simple microservice to add/upsert a `name.collate` property-of-a-property using a simple function to strip all diacritic marks - for example:

```
str.normalize("NFD").replace(/[\u0300-\u036f]/g, "").toLower()
```

Other substitutions could be made where local spelling rules vary (for example different for German ö = oe).

G.2.2 Route language sensitive queries via a proxy

Create a simple forwarding proxy around the NGSI-LD system. For any urls with a q param (and a collate flag) run a clean-up of the q param and amend the query string:

The following request on the proxy:

GET /ngsi-ld/v1/entities/?type=Building&q=name==%22Schöne%20Grüße%22&collate=name

is altered on the fly and is sent to the NGSI-LD system as shown:

GET /ngsi-ld/v1/entities/?type=Building&q=name.collate==%22schoene%20gruesse%22

Once again, the substitutions to make to the query string will depend on the rules of the natural language to be supported.

G.3 Localization of Dates, Currency formats, etc.

G.3.0 Foreword

Context data entities are designed to be interoperable and therefore all dates are held as UTC dates, all currency amounts are held as JSON numbers (with the unitCode property-of-a-property available to hold the currency), etc. Localization should not occur within the context data entities themselves. Offering fully localized responses is not a concern of the NGSI-LD API.

If localization support is necessary, a simple proxying a conversion mechanism could be used to amend the context data received from the NGSI-LD system before being passed to a third party system for display.

G.3.1 Localizing Dates

For example, if a system needs to display DateTime data in Islamic Date format

The following request on the proxy:

GET /ngsi-ld/v1/entities/urn:ngsi-ld:Event:XXX?attrs=date&options=keyValues

is forwarded unaltered and is sent to the NGSI-LD system as shown:

GET /ngsi-ld/v1/entities/urn:ngsi-ld:Event:XXX?attrs=date&options=keyValues

The response from the NGSI-LD system is always in UTC format:

```
{"date": "2020-09-28T17:13:39+02:00"}
```

And the proxy can be used to update this to the desired format:

```
    {"date": "11 Safar, 1442 1:13:39PM"}
```

Using an internationalization script such as the following:

```
new Intl.DateTimeFormat("en-u-ca-islamic", {day: 'numeric', month: 'long',weekday: 'long',year :
'numeric'}).format(date);
```

It should be noted that post-localization, the transformed date is no longer valid NGSI-LD.

Annex H (informative): Suggested actuation workflows

H.1 Actuators and feedback to the consumer

Actuators are things that can change their state (light on/off) or execute actions (move forward, detect face, etc.).

There is currently no explicitly and precisely specified support for actuation in the NGSI-LD API. Thus, this clause describes some conventions that represent a proposed best-practice about how NGSI-LD API and data models can be used for the interaction between applications and actuators represented by NGSI-LD Entities.

The conventions and approach described in this clause are not powerful enough to implement complex actuation jobs that depend on each other and, for instance, make actuation decisions conditional on the outcome of other actuations, unless that behaviour is implemented in a custom way within the application logic. The concept of a more evolved service execution logic, being a first-class citizen of the NGSI-LD API and able to offer more structured building blocks for actuation, is outside the scope of this annex.

An NGSI-LD system that comprises an actuator and supports actuation workflows is represented as one or more NGSI-LD Entities, plus a Context Broker, a Context Source or a Context Producer, and a Context Consumer, which collaborate.

The interaction between actuator and Context Consumer needs to be bidirectional. Thus, actuators are triggered by the reception of actuation-specific commands (e.g. "set the on state of the lamp to false", to turn the light off) that are encoded as NGSI-LD data, following a suggested data model. They respond with feedback, similarly encoded as NGSI-LD data.

Command feedbacks may serve to control the maximum operations rate a controlling application needs to achieve, and different levels of feedback can be requested, by associating a specific Quality of Service value to the command:

- Some applications need high operation rate but no feedback. For this case a QoS = 0 can be used. The typical example is to control the arms of a robot with a joystick.
- Some applications need to be sure that the actuators actually received the command request or need to get back a payload in response to the command. In this case a QoS = 1 can be used. The typical case is switching on a light with confirmation, or request face-detection with consequent notification of matching events.
- Commands can either require a short or a long execution time. For commands with long execution time, the application may require a continuous status feedback. In this case a QoS = 2 can be used. The typical example is that of a door opening, where feedback continuously report the current level (10 % open, 50 % open, etc.).

H.2 Architecture for actuation

In this architecture, the application acts as Context Consumer, and the terms are used interchangeably.

Commands are sent to the Context Broker by the Context Consumer, using the standard NGSI-LD API and a suggested convention for representing them. In turn, feedback about command execution is received by the Context Consumer, both as continuous status updates and/or a final command result.

More specifically, the component that handles direct communication with the actuator is the Context Source or the Context Producer: it uses an actuator-specific protocol to control the actuator and get responses and updates from it, i.e. from the real system. Such Context Source/Consumer or Context Producer/Storage acting as a proxy or intermediary to the actuator is referred to, in the following, as Context Adapter.

Thus, the Context Adapter is able to use the NGSI-LD API to receive NGSI-LD command requests from the NGSI-LD Context Broker and send back command status and result to it, as well as using an actuator-specific protocol to communicate with the actuator.

The NGSI-LD Context Broker is responsible for handling direct communication with the Context Consumer.

Thus, to support actuation, there is a need to specify:

- Additional NGSI-LD Properties the NGSI-LD system has to have, in order to represent and manage command Request, Status, Result.
- A communication model that allows commands to flow in forward direction and feedback to flow in reverse. This communication model has to comprise a mapping, to be held within the NGSI-LD system, that is able to route the command requests to the appropriate handler within the Context Adapter and vice-versa.

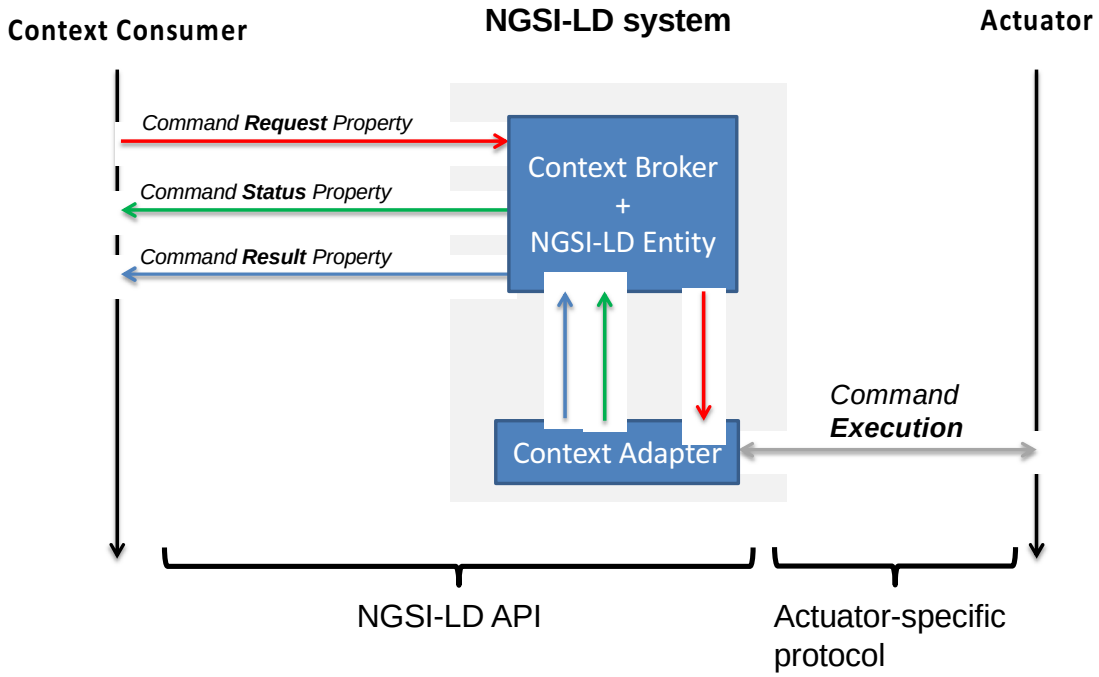

Figure H.2-1: Architecture for actuation

H.3 Structure of Commands and additional Properties

H.3.0 Introduction

The NGSI-LD system has, in addition to the usual NGSI-LD Properties representing the actuator's status, a set of additional, dedicated NGSI-LD Properties associated with:

- the list of available commands, i.e. the list of commands supported by the actuator;
- command endpoints, one for each command, that are used to send and receive command related messages and optionally hold state for the ongoing commands.

The structure of the commands needs to be specified, but not the internal format of their payloads. By using commands with a custom payload, one can support all kinds of operations, for example:

- "set-on": "true"
- "detect-face": {"face-features": "…."}
- "move": "forward"

The data model for command requests, status and responses has to include metadata such as the QoS of the command, its identifier, and the custom payload itself.

Both the requests/responses and the list of commands the NGSI-LD system is able to support can be represented with additional NGSI-LD Properties, as follows.

H.3.1 Property for listing available commands

The additional Property dedicated to the list of available commands is as follows:

```
"commands": {
  "type": "Property",
  "value": ["<cmd_name1>","<cmd_name2>", …, "<cmd_nameN>"]
}
```

It is a Property whose value is an array of Strings, each string representing the unique name of a supported command.

H.3.2 Properties for command endpoints

For each available command, a set of three endpoints is to be additionally created within the NGSI-LD system, by means of three dedicated Properties per command. The first endpoint will manage that command's requests, the second endpoint will manage its status, and the third endpoint will manage command's results.

This convention dictates that:

- The NGSI-LD Property that manages requests has the same name as the command, e.g. "<cmd_name1>".
- The NGSI-LD Property that manages results has the same name as the command plus the "-RESULT" suffix.
- The NGSI-LD Property that manages status has the same name as the command plus the "-STATUS" suffix.

Each endpoint can receive multiple requests or responses, and it supports queueing of messages. For example, the command "moveToLocation" may receive a sequence of requests that are to be stored in an array and orderly processed depending on the arrival timestamps. A number of respective responses may be produced, as well. Thus, each endpoint, represented by its dedicated NGSI-LD Property, exploits the multi-Attribute feature (see clause 4.5.5), as follows:

Command Request endpoint

```
"<cmd_name>": {
  "datasetId": a URI uniquely identifying the specific command request
               (optional, if the use case does not need command queueing),
  "type":      "Property",
  "qos":       an Integer, representing the desired QoS (optional, default=0),
  "value":     custom parameters of the command (mandatory)
}
```

Command Status endpoint

```
"<cmd_name>-STATUS": {
  "datasetId": a URI uniquely identifying the specific status feedback message
               (optional, if the use case does not need queueing them),
  "type":      "Property",
  "value":     custom status of the command (mandatory)
}
```

Command Result endpoint

```
"<cmd_name>-RESULT": {
  "datasetId": a URI uniquely identifying the specific result feedback message
               (optional, if the use case does not need queueing them),
  "type":      "Property",
  "value":     custom result of the command (mandatory)
}
```

Usually, the Context Adapter (or the actuator behind it), upon receiving a command request with a specific "datasetId", will then generate status and result with the same "datasetId", so that, when the status/result is received by the application, it can link it back to the corresponding command that is generating the received feedback. The value of the request, status and result is generic, and it is up to the specific application to define useful values. As an example, the PackML language for the control of packaging machines defines a set of possible values for statuses during an actuation workflow.

EXAMPLE 1: An example follows, where the NGSI-LD system represents a simple actuator. The example shows the NGSI-LD Entity representing a light that can change colour by manipulation of its brightness, hue and saturation values; further, it is possible to turn the lamp on and off. Apart from the **"id"** and the **"type"**, the actuator entity has a set of regular properties that represent the current status of the lamp. In the example these are "**colorRGB**" and "**is-on**". Then it has the conventional Property named "**commands**", signalling that it supports four commands: **"turn-on", "set-saturation", "set-brightness", "set-hue"**. Further, it has four (times three) additional properties serving the purpose of command endpoints.

```
{
  "id": "urn:ngsi-ld:pHueActuator:light1",
  "type": "Lamp",

  REGULAR PROPERTIES
  "colorRGB": {"type": "Property", "value": "0xABABAB"},
  "is-on": {"type": "Property", "value": true},

  AVAILABLE COMMANDS
  "commands": {
    "type": "Property",
    "value": ["turn-on", "set-saturation", "set-hue", "set-brightness"]
  }

  COMMAND ENDPOINTS
  "turn-on": {"type": "Property", "value": <custom request>}
  "turn-on-STATUS": {"type": "Property", "value": <custom status>}
  "turn-on-RESULT": {"type": "Property", "value": <custom response>}
  "set-hue": ...
  "set-hue-STATUS": ...
  "set-hue-RESULT": ...
  …
}
```

EXAMPLE 2: The following example, shows an NGSI-LD Entity Fragment that can be used as a command request to request that the lamp be turned off.

```
{
  "id": "urn:ngsi-ld:pHueActuator:light1",
  "type": "Lamp",
  "turn-on": {
    "type": "Property",
    "qos": {
      "type": "Property",
      "value": 1
    },
    "value": false
  }
}
```

EXAMPLE 3: In the following example, the value of the command request is a more complex JSON Object, to show that complex actions can be conveyed by just one request. Further, the request has an identifier that makes it possible to enqueue it, together with other request that may arrive to the same command endpoint within a timespan.

```
{
  "id": "urn:ngsi-ld:pHueActuator:light1",
  "type": "Lamp",
  "set-hue": {
    "type": "Property",
    "qos": {
      "type": "Property",
      "value": 1
    },
    "datasetId": "myapp:mycommand:1342",
    "value": {"red":"1 seconds", "green": "2 seconds"}
  }
}
```

In summary, the suggested convention prescribes a "**commands"** property that contains a list of commands supported by the actuator. For each of these commands, the convention requires a command endpoint consisting of three properties, the name of the command, e.g. "**turn-on**", the status property, which is the name of the command with "-**STATUS**" as suffix, and the result, which is the name of the command with "-**RESULT**" as suffix. Nevertheless, it is noted that such suffixes are just a convention to distinguish the endpoints. So far, two practical implementations exist, see clauses H.5 and H.6, that adopt the general scheme of this convention, with minor deviations. In fact, this convention is derived as a generalization that leverages the full potential of NGSI-LD sub-Attributes and multi-Attributes.

H.4 Communication model

H.4.1 Possible communication models

This convention can be leveraged by two different communication models:

- Subscription/notification, where both the application and the Context Adapter use NGSI-LD Subscriptions to have the command requests delivered to the appropriate handler within the Context Adapter and vice-versa. In this case the Context Adapter acts as a Context Source as well as a Context Consumer.

- Forwarding, which uses the NGSI-LD Registry and a Context Adapter able to federate itself with the Context Broker holding the actuator's Entity, as a means to deliver the commands. In this case the Context Adapter acts as a Context Storage as well as a Context Producer.

H.4.2 Subscription/notification model

For the interaction to work, the Context Adapter, acting as a proxy to the actuator, subscribes to all command properties; in example 1 of clause H.3.2, these are "set-brightness", "set-saturation", "set-hue" and "turn-on". When the application, acting as the actuation client, updates the value of a command property, the Context Adapter will receive the notification with the new value. This will be translated into the proprietary format and forwarded to the actuator using the actuator-specific protocol. The application in turn can subscribe to the command status and the result. The Context Adapter updates the status of the actuation during the execution of the command, which is primarily relevant in the case of longer-lasting actuations, and finally updates the result once the actuation has been completed. If the application has subscribed to the status and result, it will receive the corresponding notifications. Independent of the command-related properties, the status of the actuator, held within its regular properties, will be updated.

The detailed workflow is depicted in figure H.4.2-1, and can be interpreted as follows:

1) Application updates "turn-on" command Property with "value": true

2) Context Adapter gets notification of the new value true

3) Context Adapter updates "turn-on-STATUS" command Property with "value": "PENDING"

4) Application gets notification of the new value "PENDING"

5) Context Adapter updates "is-on" regular Property with value: true

6) Application gets notification with value: true

7) Context Adapter updates "turn-on-RESULT" command Property with "value": "OK"

8) Application gets notification with of the new value "OK"

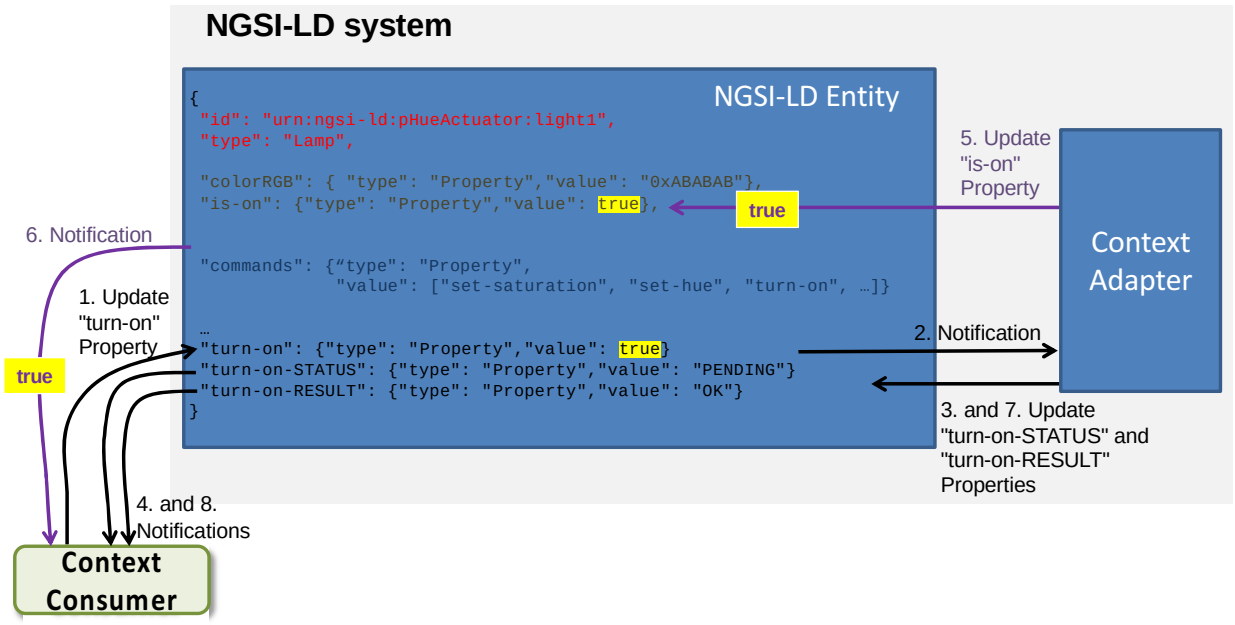

Figure H.4.2-1: Steps of the actuation workflow using subscription/notification

H.4.3 Forwarding model

The forwarding model uses registrations and forwarding of requests. Actuation of commands is provisioned via registration(s) to the NGSI-LD Registry done by the Context Adapter that states "I am responsible for command property <X>". When the Application changes the value of a command property, first the NGSI-LD Context Broker asks to the NGSI-LD Registry whether the property is delegated to some other component. The NGSI-LD Registry knows that property <X> of the Entity is delegated to the Context Adapter. Hence, the request is forwarded to the Context Adapter. Similar to the other communication model, the request will then be translated into the proprietary format and forwarded to the actuator using the actuator-specific protocol.

In this model, the NGSI-LD Entity is distributed over two different components, because some of its properties live in the Context Brokers and other properties live in the Context Adapter, as indicated in figure H.4.3-1 with a dotted rectangle.

The rest of the workflow, i.e. delivery of status and result messages to the application, is done similarly to the subscription/notification model. The detailed workflow is depicted in figure H.4.3-1, and can be interpreted as follows:

1) Application updates "turn-on" command Property with "value": true

2a) Context Broker ask Registry where to forward the request

2b) Context Broker forwards request to Context Adapter

3) Context Adapter updates "turn-on-STATUS" command Property with "value": "PENDING"

4) Application gets notification of the new value "PENDING"

5) Context Adapter updates "is-on" regular Property with value: true

6) Application gets notification with value: true

7) Context Adapter updates "turn-on-RESULT" command Property with "value": "OK"

8) Application gets notification with of the new value "OK"

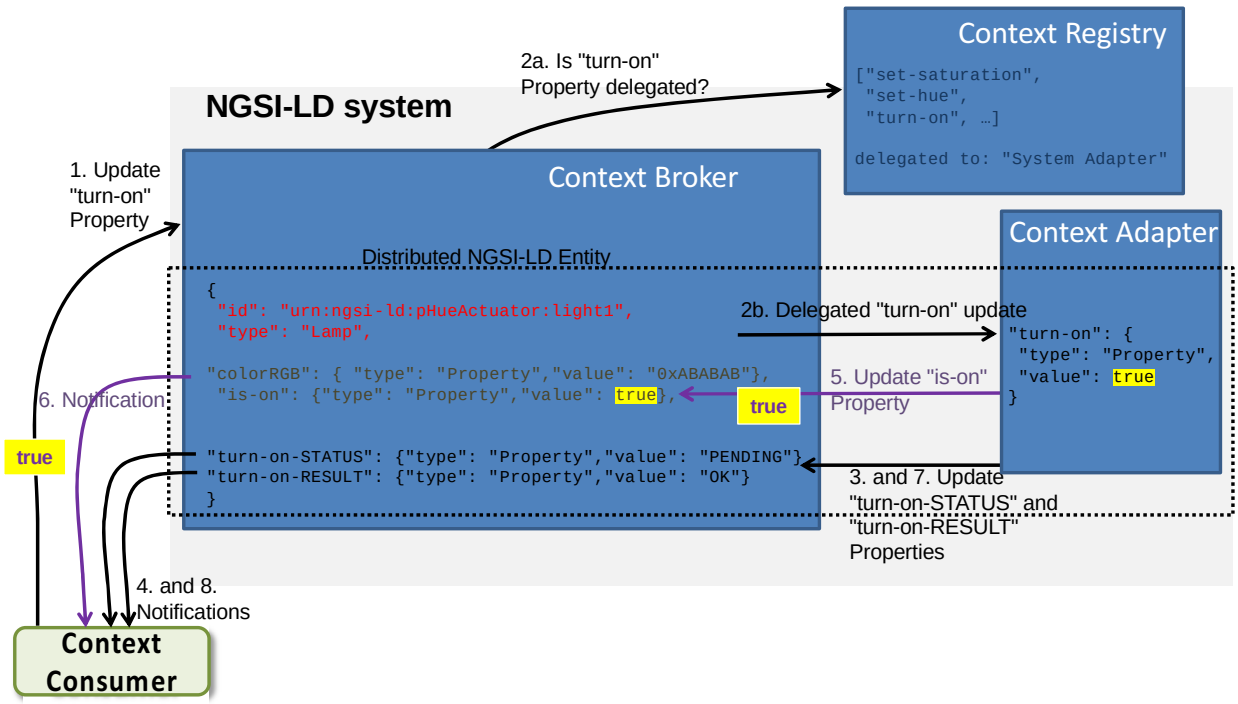

Figure H.4.3-1: Steps of the actuation workflow using forwarding

H.5 Implementation of the subscription-based actuation workflow

The Fed4IoT project (https://fed4iot.org) leverages the NGSI-LD architecture and the subscription/notification workflow for actuation, in order to implement the concept of a Cloud of Things. It enables virtualization of existing IoT sensors/actuators through Virtual Things and IoT Brokers. IoT application developers can simply rent the Virtual Things and the Brokers their applications need.

The Fed4IoT's Cloud of Things is named VirIoT (https://github.com/fed4iot/VirIoT), and it is based on the concept of Virtual Silos as-a-service: isolated and secure IoT environments made of Virtual Things whose data can be accessed through standard IoT Brokers (oneM2M, NGSI, NGSI-LD, etc.).

In figure H.5-1 a diagram shows how VirIoT implements the concept of a large-scale and distribute NGSI-LD system that leverages the architecture and the workflow convention described in clause H.4.2.

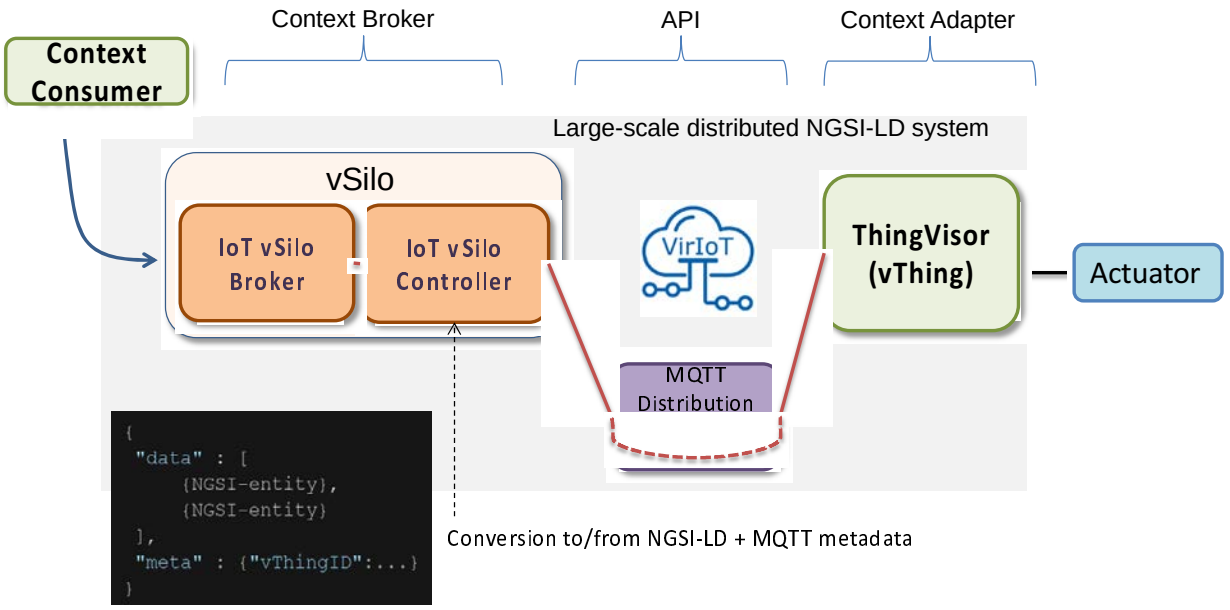

Figure H.5-1: VirIoT implementation of the NGSI-LD system and actuation workflow

All components encapsulate requests in a neutral-format message that leverages NGSI-LD Entities at its core. But, since VirIoT uses MQTT as its internal data distribution system, all information and actuation commands are encoded as NGSI-LD entities, plus an additional "meta header" that is used by the MQTT to publish and subscribe in a broadcast fashion to multiple vThings, because the same virtual sensor can be used by multiple applications at the same time.

For the actuation workflow, the "data" part of this message contains the command request, as specified in clause H.3, but with an additional value key that is the "command notification uri" (`cmd-nuri`), representing a location where feedback (status, result) should be sent by the ThingVisor. For example, the `cmd-nuri` contains the "data_in" MQTT topic of the issuing vSilo, so that command feedback (status and results) are sent to it, only, instead of being broadcasted to all subscribing applications.

VirIoT is agnostic to access control issues to a virtual actuator, since the relevant policies are implemented in the specific ThingVisor, which can grant tokens to execute actuation-commands to a subset of vSilos only, through preliminary exchange of specific actuation-commands (a kind of log-in).

Fed4IoT has developed several different ThingVisors (Context Adapters for different sensors and hardware): for example, lamp systems and robot devices are virtualized through specific ThingVisors, and applications can control the lighting system of a rented conference room or control camera and position of a bot by adding related virtual actuators to their vSilo.

H.6 Implementation of the registration-based actuation workflow

The IoT Agent node library [i.22] introduces the concept of an IoT Agent, which is a component that lets a group of devices send their data to and be managed from a Context Broker using their own native protocols. Thus, it corresponds to the Context Adapter, and wires up the IoT devices so that measurements can be read and commands can be sent using NGSI-LD requests sent to an NGSI-LD compliant context broker.

IoT Agents already exist or are in development for many IoT communication protocols and data models. Examples include the following:

- IoTAgent-JSON - a bridge between HTTP/MQTT messaging (with a JSON payload) and NGSI-LD
- IoTAgent-LWM2M - a bridge between the Lightweight M2M protocol and NGSI-LD
- IoTAgent-UL - a bridge between HTTP/MQTT messaging (with an UltraLight2.0 payload) and NGSI-LD
- IoTagent-LoRaWAN - a bridge between the LoRaWAN protocol and NGSI-LD

This implementation follows the communication model described in clause H.4.3, as explained in figure H.6-1. In this workflow:

- Requests between User and Context Broker use NGSI-LD
- Requests between Context Broker and IoT Agent use NGSI-LD
- Requests between IoT Agent and IoT Device use native protocols
- Requests between IoT Device and IoT Agent use native protocols
- Requests between IoT Agent and Context Broker use NGSI-LD

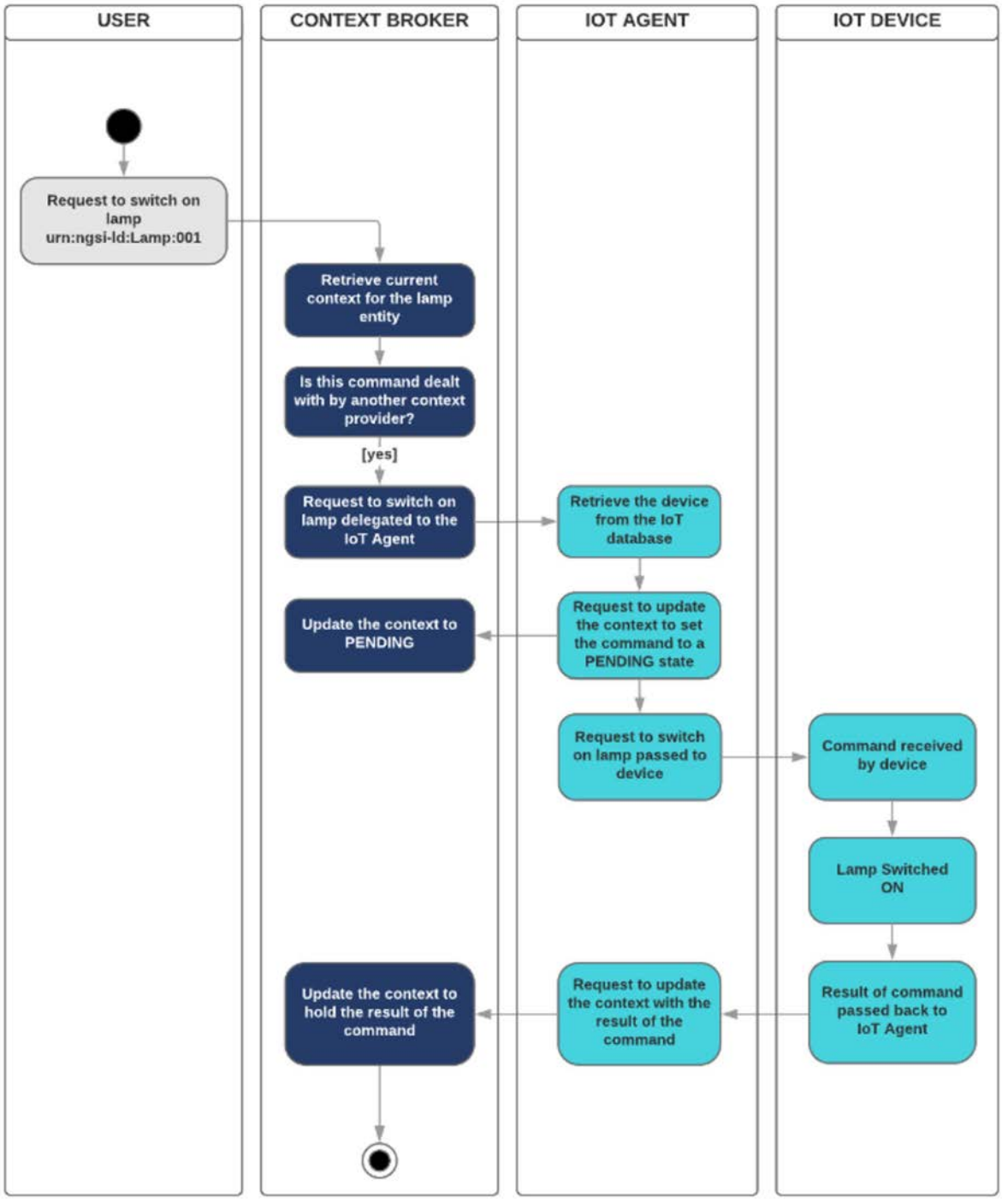

Figure H.6-1: IoT Agent node library implementation of the NGSI-LD system and actuation workflow

Provisioning of the devices will be carried out (via REST API) through IoT Agents, as well. This provisioning implies that, on the one hand, the corresponding entities (with their commands), that represent the devices, are generated in the Context Broker and, on the other hand, that the corresponding IoT Agent is configured for communication with the corresponding device, all in one provisioning step. Below, an example how to provision a device which supports start and stop commands is presented.

```
{
    "devices": [
        {
            "device_id":   "device001",
            "entity_name": "urn:ngsi-ld:Device:001",
            "entity_type": "Device",
            "attributes": [
                { "object_id": "s", "name": "isOpen", "type": "boolean" }
            ],
            "commands": [
                { "name": "start", "type": "command" },
                { "name": "stop", "type": "command" }
            ],
            "static_attributes": [
                {"name":"name", "type": "Text","value": "Device:001 provision"}
            ]
        }
    ]
}
```

Annex I (informative): Change history

Date	Version	Information about changes
February 2020	1.2.10	Early draft copied from API version 1.2.1
February 2020	1.2.11	Unicode characters. Query Language syntax changes to Attribute path, and extension to accept specifying just Query or Geoquery when Querying Entities Acknowledgements to EU projects. Lightweight Figures
March 2020	1.2.12	Extending to other interactions the above changes to query entities interaction Changes to ABNF Query Language syntax to access complex objects value of properties more easily Generalized Notification Headers, in order to carry authentication etc., info Novel &option=count and associated Header to indicate number of Entities in response to a query Novel/entityOperations/query and/temporal/entityOperations/query endpoints to perform query via POST Clarified attrs URL parameter behaviour Support for Multiple Attributes Support for Multiple Tenants
May 2020	Candidate 1.2.13	from 101r1: Multi-Attribute-Support-fix-in-4.5.5 from 102r1: Batch_Operation_Error_Codes from 110r1: JSON-LD Validation clause from 112r1: IRI Support for International Characters from 115r2: More Core Context Changes from 130: Entity Types MQTT Notifications GeoJSON Representation
9 July 2020	1.3.1	Technical Officer verifications for submission to editHelp! publication pre-processing
August 2020	1.3.2	New baseline towards v1.4.1
November 2020	1.3.3	From 272r1: Support for natural languages via LanguageProperty; annex G
December 2020	1.3.4	From 319: Align table 6.8.3.2-1 with clause 5.10.2 for query via attrs
December 2020	1.3.4	From 310r2: Dot vs. comma in DateTime
December 2020	1.3.4	From 309r1: Remove sentences referring to old multi attributes representation
December 2020	1.3.4	From 308r: id and type for JSON-LD compliance
December 2020	1.3.4	From 313r1: FIXES to Cross domain data model for LanguageProperties Bug fixes and errata
December 2020	1.3.5	From 275r3: Temporal Aggregation Functions
December 2020	1.4.0	1.3.5 with small typos corrected, approved as 1.4.0
January 2021	1.4.1	ETSI Technical Officer review for ETSI EditHelp publication pre-processing
March 2021	1.4.2	Editorial Changes, clarifications added, better references, figures replacements and corrections, figures merged, typos, code identation
April 2021	1.4.2	Temporal Pagination
April 2021	1.4.2	Clarified behavior when multiple instances of the same Entity are in an input array
July 2021	1.4.3	From 130r6: NGSI-LD Scope
July 2021	1.4.3	From 143r6: Storing, managing and serving @contexts
July 2021	1.4.3	From 120r4: API structuring
October 2021	1.4.4	From 156: Remove static elements from temporal representations
October 2021	1.4.4	From 155: Existence query
October 2021	1.4.4	From 152: Remove null value deletion
October 2021	1.4.4	From 151: attrs missing in core context
October 2021	1.5.1	ETSI Technical Officer review for ETSI EditHelp publication pre-processing
November 2021	1.5.2	First early draft after EditHelp publication of V1.5.1 to prepare next V1.6.1 publication
January 2022	1.5.3	Concise representation
May 2022	1.5.4	PUT/PATCH Entity
May 2022	1.5.4	Distributed operations
July 2022	1.5.5	From 99r6: Deletions and advanced notifications
July 2022	1.5.5	From 106r1: Workflow for actuation
July 2022	1.5.5	From 105r1: Context Source Info in Context Source Registration
July 2022	1.5.5	From 93r1: default context clarification
July 2022	1.5.5	From 91r1: CR_on_Scope_ABNF_format
Juy 2022	1.6.1	Final Technical Official check for EditHelp publication

Date	Version	Information about changes
October 2022	1.6.2	New early draft: corrected Annex C6 date representation from 149r3: generalized description of @context in bullet lists Fixed usage of NGSI-LD Null in Attributes' definitions
December 2022	1.6.4	From 188r2_Registration_Clarifications
December 2022	1.6.4	From 182r2_Add_NGSI-LD_Roles_for_Context_Registration
December 2022	1.6.4	From 156r2 Vocabulary Property/URI type coercion
December 2022	1.6.4	177r2 Clarify usage of Accept, Content-Type and Linked @context when forwarding to partial brokers
December 2022	1.6.4	178 Add Batch Query to Federation Ops
December 2022	1.6.4	183r1 Clarify Temporal query behaviour
December 2022	1.6.4	149r4 Forbid scoped and nested @contexts
December 2022	1.6.4	023006 Fixing CSource registration example in C.3
December 2022	1.6.4	Fix: Tenants URI becomes String
December 2022	1.6.4	Fix: POST-QUERY-COUNT-PAGINATION
December 2022	1.6.4	Fix: CHECK-URI-PARAM
December 2022	1.6.4	Updated examples and text to context.v1.7.jsonld
March 2023	1.6.6	CIM(23)000006_Adding_previousValue_to_GeoProperty_type_definition
March 2023	1.6.6	cSource -> CSource; "implementations shall do the following"
March 2023	1.6.7	000013r4_Updated_Distributed_Execution_Behaviour
March 2023	1.6.8	CIM(22)000195r3_type_passing_for_M2M_callReviewed
April 2023	1.6.9	Fixing URI→String datatypes

History

Document history		
V1.1.1	January 2019	Publication
V1.2.1	October 2019	Publication
V1.2.2	February 2020	Publication
V1.3.1	August 2020	Publication
V1.4.1	February 2021	Publication
V1.4.2	April 2021	Publication
V1.5.1	November 2021	Publication
V1.6.1	August 2022	Publication
V1.7.1	June 2023	Publication