

New Features and Relevant Fixes in NGSI-LD v1.5.1

Presented by: **Martin Bauer, Giuseppe Tropea,
Stefan Wiedemann and Ken Zangelin**

For: **ETSI ISG CIM**

23.11.2021

Overview

- New Features in NGSI-LD v1.5.1
 - Multi-typing for Entities
 - NGSI-LD Scope
 - Storing, Managing and Serving @context
 - Pagination of Temporal Attributes
 - Support for Different Instances of same Entity in Batch Operations
- Some Fixes in NGSI-LD v1.5.1 and their Rationale
 - NGSI-LD API Structure and Implementation Options (incl. internal temporal updates)
 - Remove Option to Delete Value by Updating with *null* Value
 - Remove Underspecified Option of Returning Static Values in Temporal API
 - Clarify Semantics of Existence check in Filter Query

[CIM 009v1.5.1] https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.05.01_60/gs_CIM009v010501p.pdf

New Features in NGSI-LD v1.5.1

Multi-Typing for Entities (1)

- Previous versions of NGSI-LD (up to NGSI-LD v1.4.2) allowed entities to have **exactly one entity type**
- With **Multi-Typing**, entities can have **more than one entity type** at the same time

Example:

- A 'motorhome' can be a 'home' and a 'vehicle' at the same time, i.e. such an entity could have the following entity types:
 - motorhome
 - home, vehicle
 - motorhome, home, vehicle
 - The latter has the advantage that the entity would be found when looking for any of the entity types
- **Note:** *Subtyping would be another general modelling option ('motorhome' as a subtype of both 'home' and 'vehicle') – however, to find such subtypes, NGSI-LD Brokers would be required to know the type hierarchy (as specified by a data model), and the current assumption is that NGSI-LD Brokers are agnostic to data models.*

Multi-Typing for Entities (2)

- Changes:
 - Entity Type has cardinality 1..N, so, in line with JSON-LD, either a **single type** or a **JSON array with multiple types** will be provided → as long as only a single type per entity is ever used, nothing changes for the user!
- Queries and Subscriptions
 - Queries and subscriptions already allow providing a comma-separated list of entity types when specifying entities of interest: `type=home, vehicle`, which corresponds to a **disjunction** (home OR vehicle)
 - in addition, **conjunctions of entity types** are required (home AND vehicle)
- **Decision 1:** support **disjunctions** where elements can either be **single entity types** or **conjunctions of entity types**
- **Decision 2:** use **conjunction operator ‘;’** and **disjunction operator ‘|’** as in the query language, but for compatibility reasons, **allow ‘,’ as alternative disjunction operator**
- **Example:** `type=motorhome | (home; vehicle)` – alternative:
`type=motorhome, (home; vehicle)`

Multi-Typing for Entities (3)

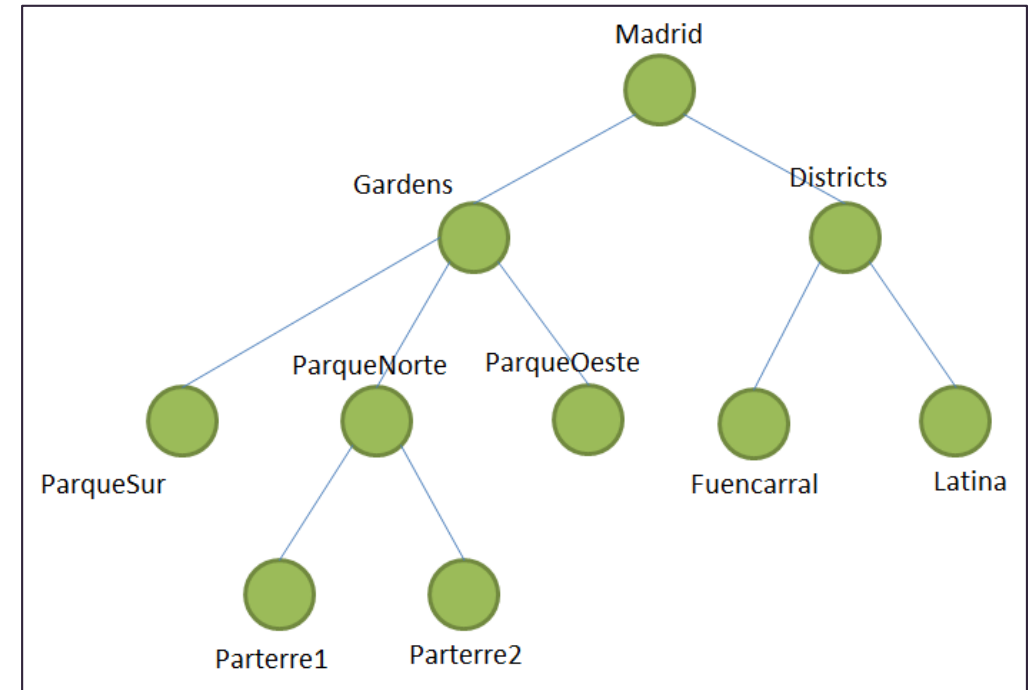
- So far ***entity type*** has been **immutable**, i.e. once the entity was created it could not be changed (only option: first delete entity and then recreate it with new entity type).
- **Decision: update and append operations can, implicitly, add new entity types, but existing entity types are not removed**
 - Reason: if removing entity types was allowed this could lead to unexpected behaviours

Options to change entity type

- Delete and re-create, as previously
- Batch upsert with “replace” option: “Indicates that all the existing Entity content shall be replaced”

NGSI-LD Scope (1)

- **Hierarchical structures** play an important role for **structuring and organizing our world**, e.g. hierarchical location structures (example on the left) or organizational structures (e.g. of a company)
- The special ***scope*** property allows giving an entity a hierarchical scope, e.g. *ParqueSur* in the example on the left
- In queries and subscriptions, scopes can be used for filtering with the ***scopeQ*** parameter, e.g. *scopeQ="/Madrid/Gardens/ParqueSur"*, which would only match entities that have this particular scope.



[CIM 009v1.5.1] Clauses **4.18**, **4.19**, 4.5.1, 4.5.6, 5.2.4, 5.2.12, 5.2.23, 6.4.3.2, 6.8.3.2, 6.18.3.2, C.5.15, C.5.16




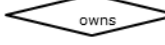
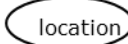
NGSI-LD Scope (2)

- Entities can have **multiple** hierarchical scopes attached, e.g.

```
"scope": [
    "/Madrid/District/Latina",
    "/CompanyA/DepartmentX/UnitC"
]
```
- Scopes are **optional** and **independent of the Entity ID**
- Scopes can be used to **scope queries and subscriptions**
- The **scope query language** (scopeQ) allows **wildcards**
 - '+' for an arbitrary value of one hierarchy level, e.g. *scopeQ="/CompanyA/+/HR"*, for matching the HR groups of all departments of CompanyA
 - '#' matches the given scope and the whole hierarchy of scopes below, e.g. *scopeQ="/Madrid/Gardens/#"* matches all entities with scope */Madrid/Gardens* or any of its direct or indirect sub-scopes
 - '/#' matches all entities that have any explicit scope attached
- The **scope query language** (scopeQ) has logical *and* and *or* operators, e.g. *scopeQ="/Madrid/District/Centro,/Madrid/District/Cortes"* for all entities whose scope is */Madrid/District/Centro* or */Madrid/District/Cortes*
- Scopes of entities **can be updated**
- Scopes can **be registered as part of registrations**

Storing, Managing and Serving JSON-LD @context (1)

- Background: NGSI-LD is encoded in JSON-LD and JSON-LD maps short names to unique URIs
 - This mapping is specified in an @context.
 - The @context can be embedded in the JSON document (for *create* and *update only*)
 - The @context can be fetched from a URL put into an HTTP Link Header (all cases, **preferred**)
- User generated @context is hosted somewhere
- The Context Broker downloads it (at every API call!)
- For efficiency, Context Broker **needs to cache @context**
- It would be good if, additionally, the Context Broker could **host @contexts** on behalf of users

Person 	https://forge.etsi.org/gitlab/exampleOntology/Person
Vehicle 	https://forge.etsi.org/gitlab/exampleOntology/Vehicle
Shop 	https://forge.etsi.org/gitlab/exampleOntology/Shop
 owns	https://forge.etsi.org/gitlab/exampleOntology/owns
 location	https://forge.etsi.org/gitlab/exampleOntology/location

Storing, Managing and Serving JSON-LD @context (2)

The new API allows:

- storing, managing and serving custom user JSON-LD @contexts
- Automatic caching of downloaded @contexts

Specifically:

- Add a user-defined @context
- List both the automatically cached and the explicitly stored @contexts
- Serve a user-defined @context / Get info about a cached @context
- Delete a cached or stored @context
 - in case of cached @context, reloading a fresh copy can be forced, deleting only if successful

Pagination of Temporal Attributes

Retrieval of temporal representations can become very resource consuming, thus a new mechanism for pagination in such cases was added:

- The implementations should implement a default limit for the retrieval of temporal representations
- In case the limit is exceeded, a "Partial Content (206)" status will be returned
- The "Content-Range" header will return information about the responded time-range (start, end, size)
- The solution complies with the IETF RFC 7233

[CIM 009v1.5.1] Clause **6.3.10**

Support for Different Instances of same Entity in Batch Operations (1)

It is common for user applications to buffer modifications of entities and send a batch of updates (as an array of entities) to the NGSI-LD broker every X seconds.

If one and the same entity is updated more than once in a batch, then the broker has a slight problem. What if an attribute A of the entity Y is set to 5 in one update, and set to 11 in another?

In order to fix this problem, an NGSI-LD broker assumes the entity array of a batch operation is ordered chronologically, with lower array index meaning prior in time.

Support for Different Instances of same Entity in Batch Operations (2)

For the “current state”, all updates are taken in order (lower array indices are treated first), to compose the new state of the entity, working the exact same way as if the updates would have entered the broker separately and in the order they come in the batch array.

For the Temporal Representation of the Entity, each and every update is added to the temporal database. Unfortunately, as `createdAt` and `modifiedAt` are automatic attributes, set by the broker, all updates get the exact same timestamps. It is highly recommended to make use of the “`observedAt`” sub-attribute for the attributes, to preserve the time-instances in which the updates were made.

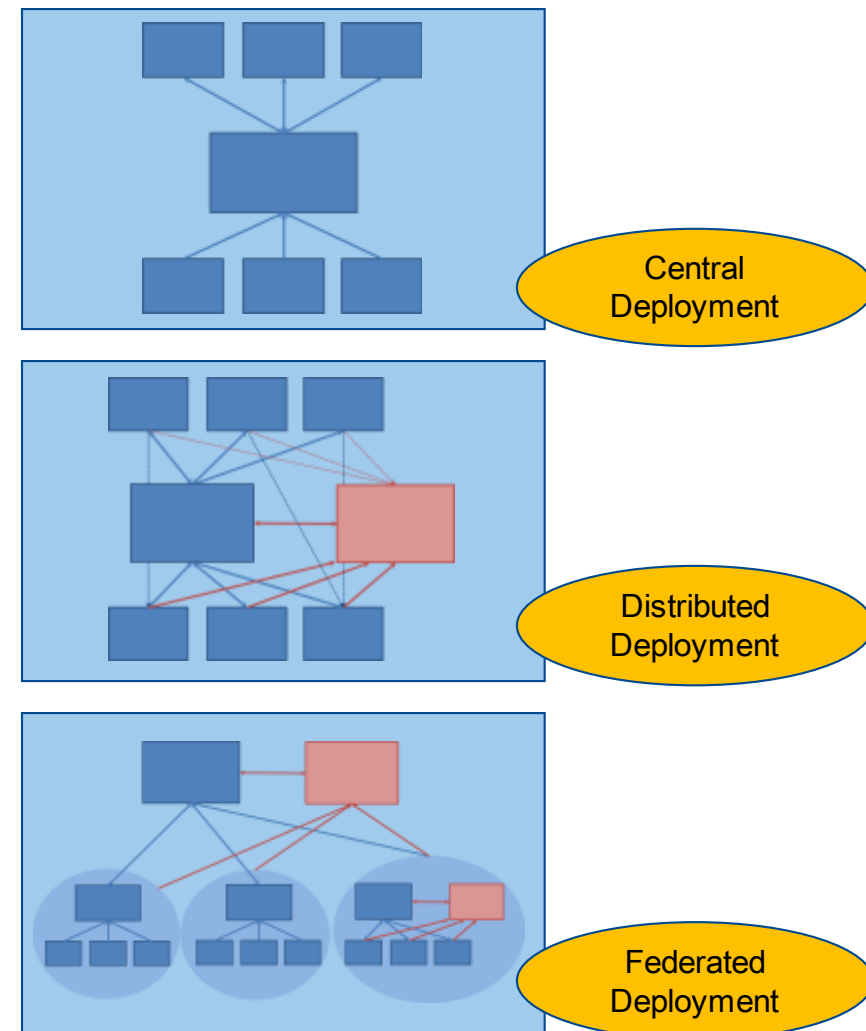
Batch operations deal with creation, modification, and deletion of entities, so the way the batch array is treated depends on the operation. For all operations, it works **the exact same way as if the operations would have entered the broker separately and in the order they come in the batch array.**

For example, if it's a DELETE operation, the first instance of an entity will actually delete the entity, while the subsequent instances will do nothing (except provoking a 207 Multi-Status response).

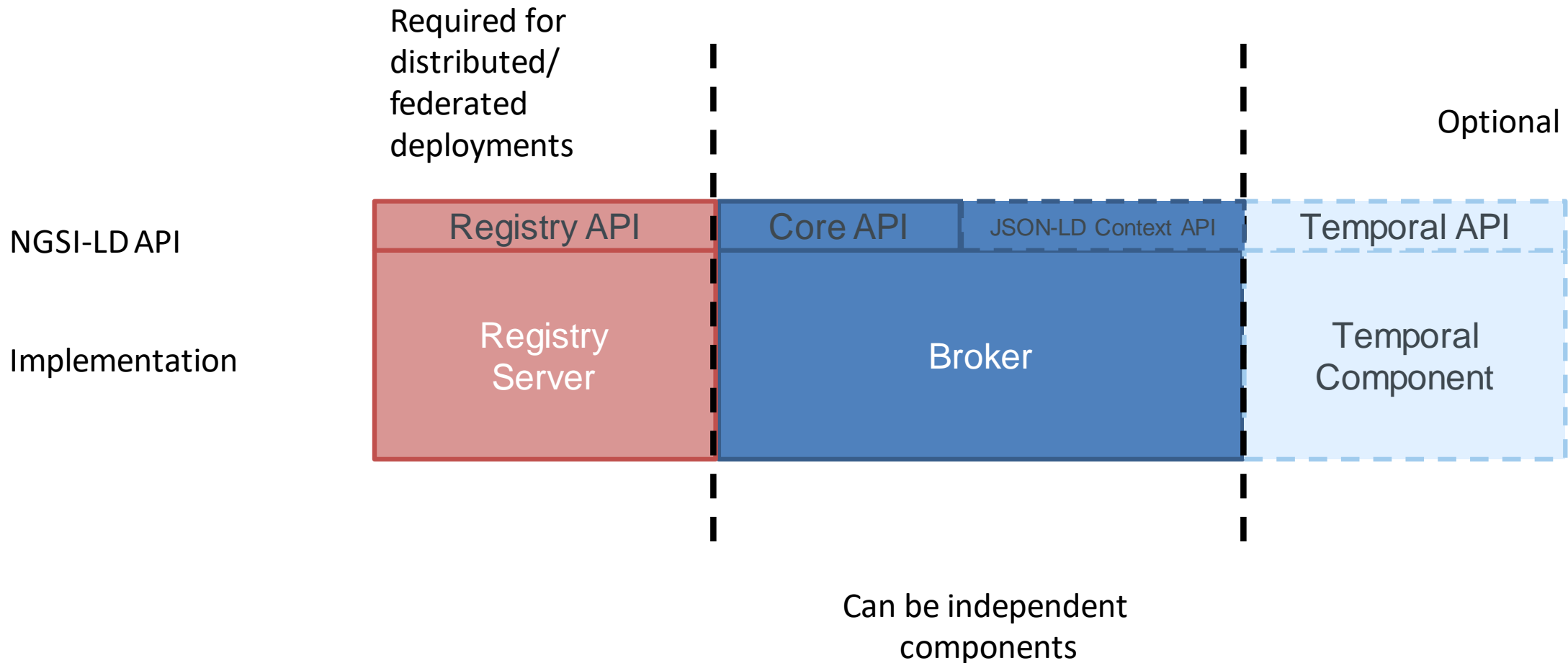
Some Fixes in NGSI-LD v1.5.1 and their Rationale

Clarification in NGSI-LD Specification: NGSI-LD API Structure and Implementation Options (1)

- NGSI-LD API can be used in multiple architectures
- Depending on architecture, not all aspects of the NGSI-LD API have to be implemented
- Brokers: All parts of the NGSI-LD API can be implemented by a single Broker OR some parts of the API can be implemented by independent components



NGSI-LD API Structure and Implementation Options (2)



NGSI-LD API Structure and Implementation Options (3)

API	Functionality	Clauses	
Core API	Context Information Provision	5.6.1 Create Entity 5.6.2 Update Entity Attributes 5.6.3 Append Entity Attributes 5.6.4 Partial Attribute Update 5.6.5 Delete Entity 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
	Context Information Consumption	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
	Context Information Subscription	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 Information Provision	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 Information Consumption	5.7.3 Retrieve Temporal Evolution of Entity 5.7.4 Query Temporal Evolution of Entities	

NGSI-LD API Structure and Implementation Options (4)

API	Functionality	Clauses	
Registry API	Context Source Registration	5.9.2 Register Context Source 5.9.3 Update Context Source Registration (CSR)	5.9.4 Delete CSR
	Context Source Discovery	5.7.1 Retrieve CSR	5.7.2 Query CSRs
	Context Source Registration Subscription	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

Recommendation:

Brokers that implement an internal Temporal Component with Temporal API should consider updating the Temporal Evolution of an Entity whenever the "current state" is modified via the Core API.

Remove Option to Delete Value by Updating with *null* Value

- Previous versions of the API did foresee that attribute values can be deleted as part of an update by assigning the value *null*.
- As it turned out, this is not compatible with the way *null* values are handled in JSON-LD. If the value of an element is set to *null* in a JSON-LD document, it will be removed completely in a JSON-LD expansion operation.
- Many NGSI-LD Brokers use standard JSON-LD expansion when receiving an NGSI-LD request, i.e. all elements with *null* values are immediately discarded and thus they are no longer there to trigger the deletion of the element from the internal database.
- As there is no way to properly handle this in JSON-LD, it was decided to remove this option from NGSI-LD, i.e. attributes have to be deleted explicitly with a delete operation.

Remove Underspecified Option of Returning Static Values in Temporal API



It was possible to have, in the temporal representation, attributes **without** a timestamp

- This was known as a "static" temporal attribute (i.e. it has not changed over time)

However the details of "static" temporal attributes were not clearly specified

- E.g. how are such attributes added?

Hence, support for "static" temporal attributes **has been removed**

- Only temporal attributes with timestamp are now allowed in the temporal representation of an entity

Removed from [CIM 009v1.5.1] Clause **4.5.7, 4.5.8, 4.5.9**, 5.7.3, 5.7.4

Clarify Semantics of Existence check in Filter Query

The NGSI-LD query language allows filtering entities according to the existence of attributes. The syntax for this existence query was already allowed in previous releases of the NGSI-LD query language, but the semantics was not defined.

For example, **GET /entities?q=attrX** will return only those entities that have an attribute called **attrX** (after expansion, of course), regardless of the value of said attribute.

If you instead want to filter on a value of an attribute:

GET /entities?q=attrX==12