New Features and Relevant Fixes in NGSI-LD v1.6.1

Martin Bauer

ETSI ISG CIM

28/09/2022

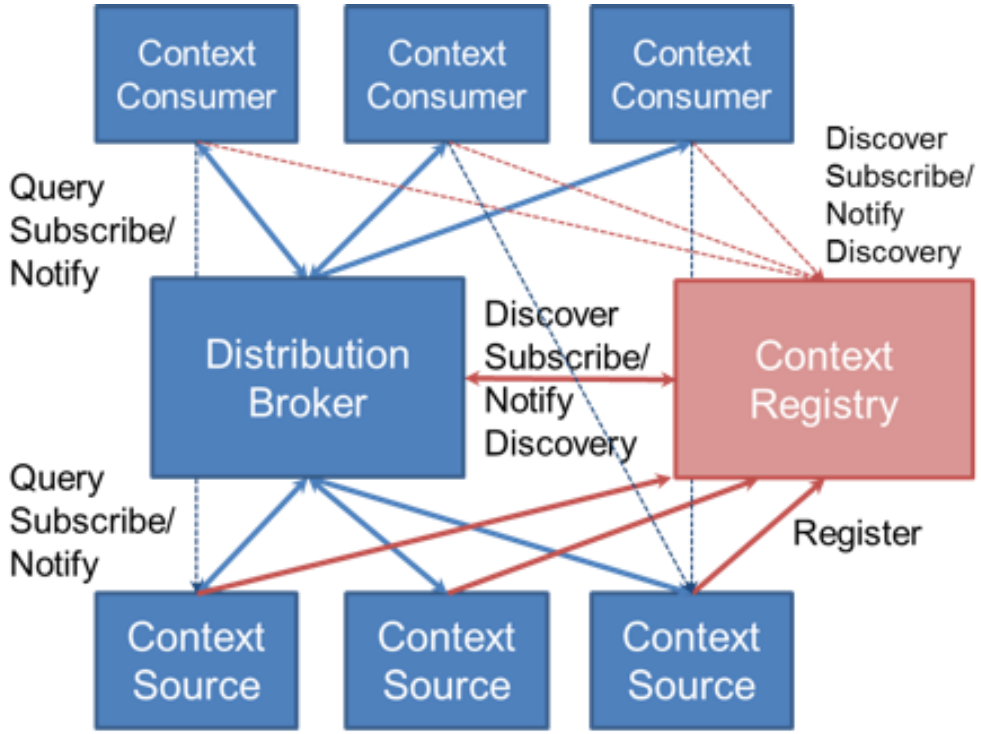
Overview

- New Features and Fixes in NGSI-LD v1.6.1
  - Distributed Operation
  - Concise NGSI-LD Representation
  - Advanced Notification Conditions
  - Representing deleted Entities and Attributes in Notifications and Temporal Evolution
  - Merge/Patch and Replace Operations (including deletions)
  - contextSourceInfo added to Context Source Registration
  - Actuation workflow
  - Clarification on use of a default @context
  - Minor change to Scope representation

[CIM 009v1.6.1] https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.06.01_60/gs_CIM009v010601p.pdf

Distributed Operation (1)

- Distributed operation has been considered from the beginning, but there are different scenarios that require different features
  - Not all registered NGSI-LD components implement the full set of NGSI-LD operations, thus the operations supported by the Context Source need to be specified
  - Depending on assumptions, attributes of the same Entity may appear in different places (e.g. *federation*) or should be exclusive to one component (e.g. *to enable actuation*)



[CIM 009v1.6.1] Clauses **4.3.6**, **4.20**, **5.2.9**, 5.2.34, **5.5.6**, **6.3.17**, **6.3.18**, 6.4.3, 6.5.3, 6.6.3, 6.7.3, 6.14.3, 6.15.3, 6.16.3, 6.17.3 (**boldface** indicates the most relevant changes for this feature)

Distributed Operation (2)

Distributed Operation Names and Groups

Individual Operations

	Operation name	Implements
Context Information Provision	createEntity	5.6.1 Create Entity
	updateEntity	5.6.2 Update Entity Attributes
	appendAttrs	5.6.3 Append Entity Attributes
	updateAttrs	5.6.4 Partial Attribute update
	deleteAttrs	5.6.5 Delete Entity 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
	updateAttrsTemporal	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

Pre-defined Groups

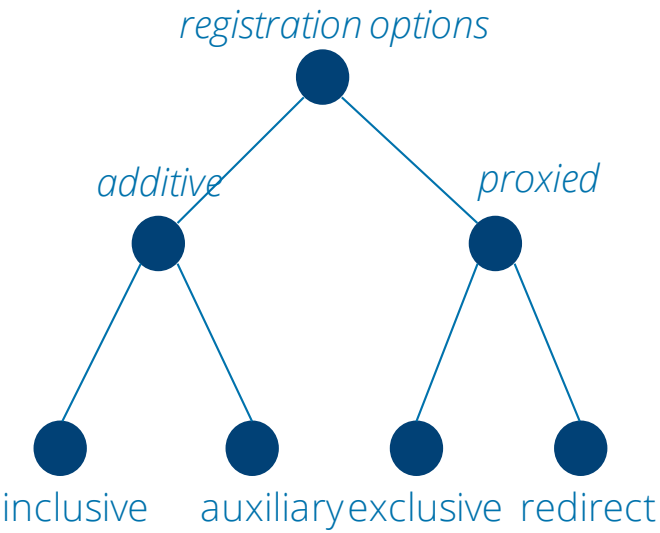
Operation Group name	Implements
federationOps	• retrieveEntity • queryEntity • retrieveEntityTypes • retrieveEntityTypeDetails • retrieveEntityTypeInfo • retrieveAttrTypes • retrieveAttrTypeDetails • retrieveAttrTypeInfo • createSubscription • updateSubscription • retrieveSubscription • querySubscription • deleteSubscription
updateOps	• updateEntity • updateAttrs • replaceEntity • replaceAttrs
retrieveOps	• retrieveEntity • queryEntity

Operation Group name	Implements
redirectionOps	• createEntity • updateEntity • appendAttrs • updateAttrs • deleteAttrs • deleteEntity • mergeEntity • replaceEntity • replaceAttrs • retrieveEntity • queryEntity • retrieveEntityTypes • retrieveEntityTypeDetails • retrieveEntityTypeInfo • retrieveAttrTypes • retrieveAttrTypeDetails • retrieveAttrTypeInfo

Operations are listed in *optional "operations"* parameter in Context Source Registration (see slide 7), default is "federationOps".

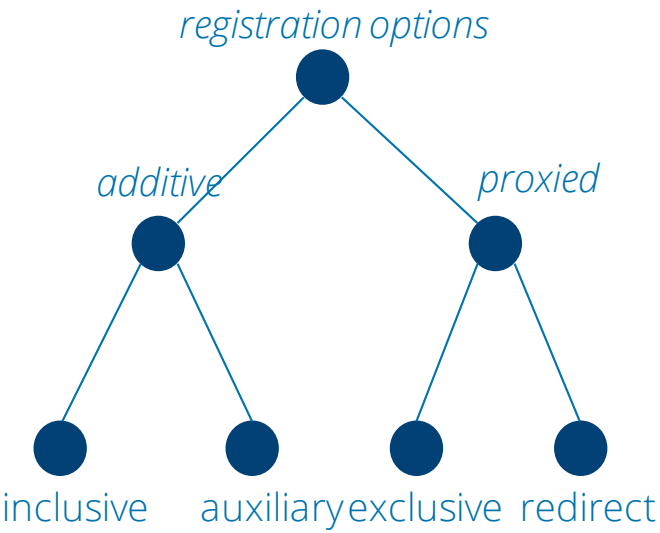
Distributed Operation (3)
Operation Modes (1)

- Additive Registrations
  - A Context Broker is **permitted to hold context data about the Entities and Attributes locally itself**, and also obtain data from (possibly multiple) external 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). 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*.



Distributed Operation (4)
Operation Modes (2)

- Proxied Registrations
  - A Context Broker is **not permitted to hold context data about the Entities and Attributes locally itself**. All context data is obtained from the external registered sources.
  - 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 **must be fully specified. It always relates to specific Attributes found on a single Entity**.
  - A **redirect** Context Source Registration also specifies that the registered context data is held in a location external to the *Context Broker*, but **potentially multiple distinct *redirect* registrations can apply at the same time**. The *Context Broker* itself cannot hold any data locally in conflict to the registration.



Distributed Operation (5)
Context Source Registration Updates

Clause 5.2.9, Table 5.2.9-1 ContextRegistration data type definition

				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 4.3.7) supported by the registered Context Source
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

Distributed Operation (6)
Limiting Distributed Operations

In order to avoid cascading distributed operations, the query parameter ***local*** can be used to limit all operations to be executed on locally available information only. (Clause 6.3.18)

Context Sources can put *localOnly* in the *management* info (Clause 5.2.34), provided in the Context Source Registration to indicate that, for all requests forwarded to it, ***local*** should be set.

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)

Concise NGSI-LD Representation (1)

- **Previously there were the "normalized" and "simplified"/"key value" representations.**

```
"isParked": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:OffStreetParking:Downtown1",
            "observedAt": "2017-07-29T12:00:04Z"
}
```

```
"isParked": "urn:ngsi-ld:OffStreetParking:Downtown1"
```

- **"normalized" is somewhat verbose and "key value" is lossy, e.g. you cannot distinguish properties and relationships if the value is a URI**
- **Some information in normalized form is redundant, as it can be inferred, so a less verbose representation can be used → "concise" ("provide only as much information as necessary")**

```
"brandname": "Mercedes"  → Property (if there are no sub-attributes)

"isParked": { "object": "urn:ngsi-ld:OffStreetParking:Downtown1" } → Relationship
                                                                     (as it has object)
"website": "http://www.fiware.org"              → Property

"availableSpotNumber": {                        → Property (with sub-attribute)
            "value": 121,
            "observedAt": "2017-07-29T12:05:02Z"
}
```

```
"location": {        → GeoProperty
            "type": "Point",
            "coordinates": [
                        -8.5,
                        41.2
            ]
}
```

[CIM 009v1.6.1] Clauses **4.5.2.3**, **4.5.3.3**, **4.5.18**, **4.7.3**, 5.2.5, 5.2.6, 5.2.7, 5.2.14, 5.2.32, **6.3.7**, C2.2, C2.3
(**boldface** indicates the most relevant changes for this feature)

Concise NGSI-LD Representation (2)

- **"Concise representation"** can be used in all places **"normalized representation"** (the primary NGSI-LD representation so far) is used
- For backward compatibility reasons, **"normalized representation" continues to be the default representation**
- If users want **concise representation** to be returned, **`options=concise`** has to be set as a URL parameter

- Users can also provide "intermediate" more verbose representations, e.g.

```
"brandname": {"value": "Mercedes"}
```

instead of simply

```
"brandname": "Mercedes"
```

- NGSI-LD components like Brokers or Context Sources are expected to **always return the (most) concise representation** if requested

- **"simplified"/"key values" representations are now also allowed for update operations. The *types* of Attributes are assumed to stay the same**, i.e. whether a URI value is the value of a Property or the object of a Relationship depends on the type of the existing Attribute being updated

Advanced Notification Conditions (1)
Previous Situation (v1.5.1 and before)

- On a (value-based) subscription, the subscriber is notified in case
  - a specified entity (based on *id*, *idParam* or *type*) with an attribute (specified in *watchedAttributes* or any attribute if not present) is ***created*** [either *type* or *attribute* have to be present]
  - an attribute of an entity (specified as above) is ***created*** / **updated**.
- There is no notification in case an entity or attribute is ***deleted***.
- You cannot select to be notified **only** in case of ***creation*** or **only** in case of ***update***.

→ **Goal: enable more flexible notification conditions to be notified in case of creation, update or deletion of entities and attributes – or any combination of these.**

[CIM 009v1.6.1] Clauses **5.2.12, 5.8.6,** 4.5.2, 4.5.3, 4.5.18, 5.2.5, 5.2.6, 5.2.14, 5.2.32, **Annex B**
(**boldface** indicates the most relevant changes for this feature)

Advanced Notification Conditions (2)

- Add optional *notificationTrigger* attribute has been added to *Subscription*, which can have one or more of the following values for Entities and Attributes respectively:

Entity	Attribute
• *entityCreated*	*attributeCreated*
• *entityUpdated*	*attributeUpdated*
• *entityDeleted*	*attributeDeleted*

- Default is *attributeCreated*, *attributeUpdated* which reflects the current semantics

Clause 5.2.12, Table 5.2.12-1 Subscription data type definition

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	0..1	Indicates that a notification shall be

©ETSI 2022 – All rights reserved

Deleted Entities and Attributes

- In order to notify about deleted entities and attributes, a representation for them is needed, which has to be defined

- Such an explicit representation is also needed in the temporal evolution of entities / attributes

  In previous API versions, such entities and attributes were simply not updated anymore, but there was no indication whether they had been deleted or just had not changed

→ This situation was completely unsatisfactory and had to be changed

[CIM 009v1.6.1] Clauses **4.5.0**, 4.5.2.2, 4.5.2.3, 4.5.3.2, 4.5.3.3, 4.5.6, 4.5.7, 4.5.8, 4.5.18.2, 4.5.18.3, 4.6.5, **4.8**, 4.18, 5.2.1, **5.2.2**, 5.2.14.1, **5.2.21**, **5.5.4**, **5.5.8**, **5.5.12**, **5.6.2**, **5.6.3, 5.6.4, 5.6.2, Annex B** (**boldface** indicates the most relevant changes for this feature)

Representing Deleted Entities / Attributes in Notifications (1)

- A *deletedAt* temporal property is added

→ In case a whole entity is deleted, only *id*, *type* and *deletedAt* temporal property is notified

- In case an attribute is deleted, an *NGSI-LD Null* value is used as a value.

  "urn:ngsi-ld:null" was chosen as it works as a *value* of a Property as well as an *object* of a Relationship

  in case of a Language Map, *{"@none": "urn:ngsi-ld:null"} is to be used*

©ETSI 2022 – All rights reserved

Representing Deleted Entities / Attributes in Notifications (2)

- Examples:

```
"brandname": {
        "type": "Property",
        "value": "urn:ngsi-ld:null"
}
"owner": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:null"
}
"street": {
        "type": "LanguageProperty",
        "languageMap": {"@none": "urn:ngsi-ld:null"}
}
```

- For deletions as part of Updates (see merge/patch slide), simply the key-value representation can be used.

```
"brandname": "urn:ngsi-ld:null"
"owner": "urn:ngsi-ld:null"
"street": "urn:ngsi-ld:null"
```

©ETSI 2022 – All rights reserved

Representing Deleted Entities / Attributes in Notifications (3)

- In the typical case only the new value – or the NGSI-LD Null in case of a deletion – is provided
- This makes sense for use cases, where the subscriber is tracking the entity/attributes and thus is aware of the previous value
- There may be some use cases, where this is not the case or does not make sense
- For such cases, it would generally make sense to provide the current value, as well as the previous value, in case there was one (otherwise it was a creation)

→ Use a special "*showChanges*" option to include both values (see following slides)

©ETSI 2022 – All rights reserved

Representing Deleted Entities / Attributes in Notifications (4)

Deletion

```
{
        "id": "urn:ngsi-ld:Vehicle:A4567",
        "type": "Vehicle",
        "deletedAt": "2022-06-03T14:47:02Z"
        "brandName": {
                "type": "Property",
                "value": "urn:ngsi-ld:null",
                "previousValue": "Mercedes",
                "datasetId": "urn:ngsi-ld:Property:manufacturer:brandname"
        }
…
}
```

- Value `"urn:ngsi-ld:null"` indicates attribute deletion.
- Additional presence of *deletedAt* attribute indicates that the whole entity was deleted

©ETSI 2022 – All rights reserved

Representing Deleted Entities / Attributes in Notifications (5)

Create / Update

```
{
        "id": "urn:ngsi-ld:Vehicle:A4567",
        "type": "Vehicle",
        "brandName": {
                "type": "Property",
                "value": "Mercedes-Benz",
                "previousValue": "Mercedes",
                "datasetId": "urn:ngsi-ld:Property:manufacturer:brandname"
        },
        "speed": {
                "type": "Property",
                "value": 55
        },
…
}
```

- Presence of (non-null) *previousValue* indicates that the attribute was updated
- Absence of previousValue indicates that attribute was created

©ETSI 2022 – All rights reserved

Representing Deleted Entities / Attributes in Temporal Evolution

- Use the *deletedAt* temporal property for deleted attributes
- Set the value of deleted Attributes to *NGSI-LD Null*
- In case a whole entity is deleted, the *deletedAt* temporal property is set for the whole entity and also all attributes are set to *NGSI-LD Null* and *deletedAt* temporal sub-property is set for each attribute

©ETSI 2022 – All rights reserved

Merge/Patch, Append and Replace Operations (1)

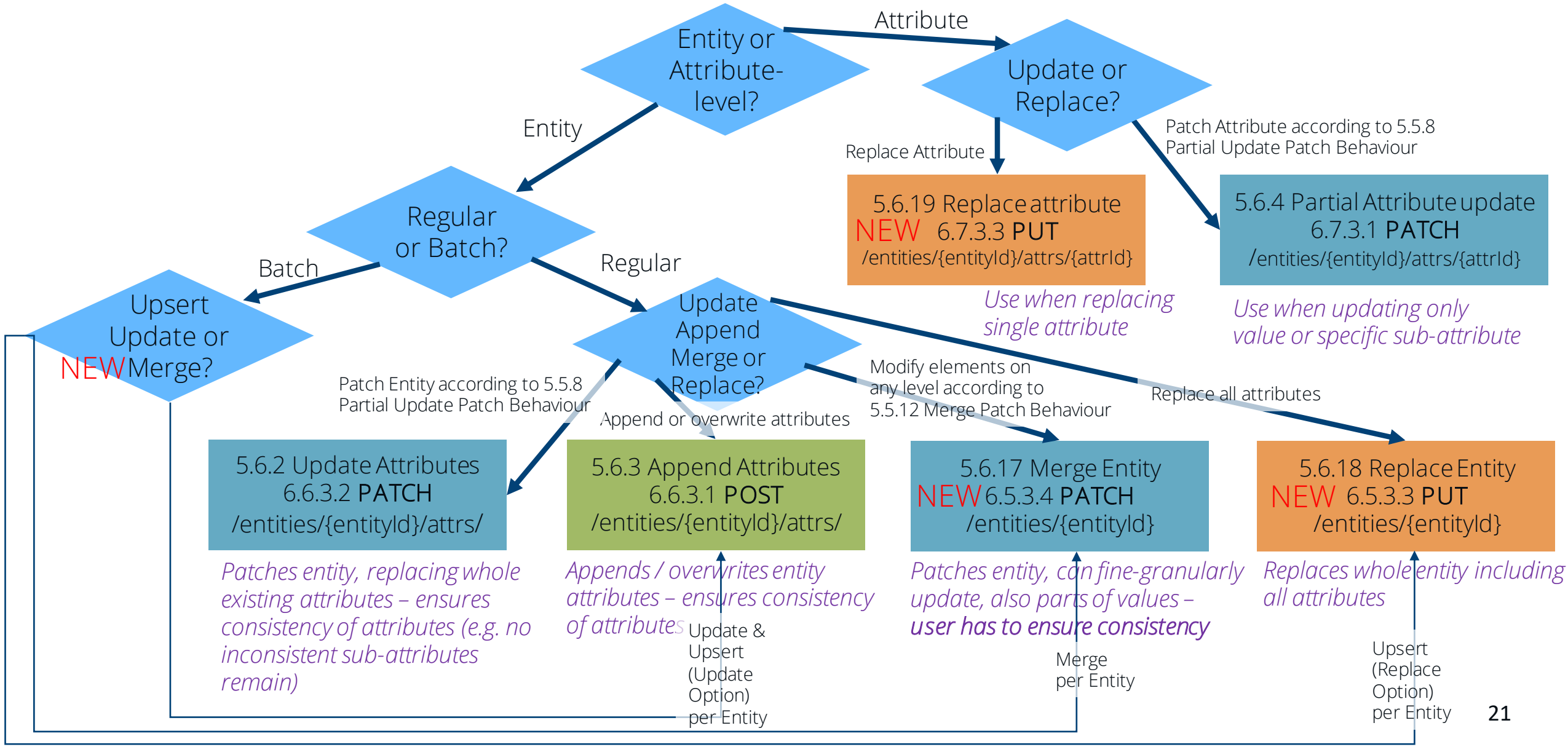
- NGSI-LD already had the following operations for updating and appending entities and attributes
  - *5.6.2 (Update Attributes)*
  - *5.6.3 (Append Attributes)*
  - *5.6.4 (Partial Attribute Updates)*
  - *5.6.7 (Batch Entity Update)*

- NGSI-LD already referred to IETF **RFC7396** for Merge-Patch Behaviour https://datatracker.ietf.org/doc/html/rfc7396, but did not completely follow it. Instead the semantics is a **mix of PATCH and PUT** (Clause 5.5.8).

- The idea is to **complement the existing operations with clear Replace (PUT) and Merge/Patch (PATCH) operations with the expected functionality** (see (new) operations on following slides)

[CIM 009v1.6.1] Clauses **5.5.8**, **5.5.12**, **5.6.2**, **5.6.3**, **5.6.4**, **5.6.7**, **5.6.17**, **5.6.18**, **5.6.19**, **5.6.20**, 6.5.3.3, 6.5.3.4, 6.7.3.1, 6.7.3.3, 6.6.3.1, 6.6.3.2

Merge/Patch, Append and Replace Operations (2)



Merge/Patch, Append and Replace Operations (3)

- In merge/patch operations, the user is always responsible for data consistency, unlike for existing partial update operations

- We also had the problem that we **could not delete as part of a PATCH by setting an attribute to *null***, as JSON-LD expansion removes anything that is set to *null*
- This is now circumvented by using the special **NGSI-LD Null** value *"urn:ngsi-ld:null"* (or *{"@none": "urn:ngsi-ld:null"}* in the case of a Language Map)
- NGSI-LD NULL for deletion is supported by 5.6.2 (Update Attributes), 5.6.4 (Partial Attribute Updates) and 5.6.17 (Merge Entity)

→ NGSI-LD v1.6.1 offers more options for updating and the behaviour better aligned with IETF RFC7396

The existing operations remain largely unchanged, but there have been some clarifications in 5.5.8 (Partial Update Behaviour), 5.6.2 (Update Attributes), 5.6.3 (Append Attributes) and 5.6.4 (Partial Attribute Updates) to address some inconsistencies. So implementations should be checked.

contextSourceInfo added to Context Source Registration

In analogy to receiverInfo in Subscription, contextSourceInfo can be specified in Context Source Registrations where additional information to be provided when contacting the Context Source can be added.

In the HTTP binding, this is translated into Headers.

You cannot overwrite information already specified elsewhere, e.g. tenant, or protocol-specific aspects like Content-length in HTTP

Clause 5.2.12, Table 5.2.12-1 ContextRegistration data type definition

				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	Scope(s)	0..1	Scopes (see clause 4.18) for

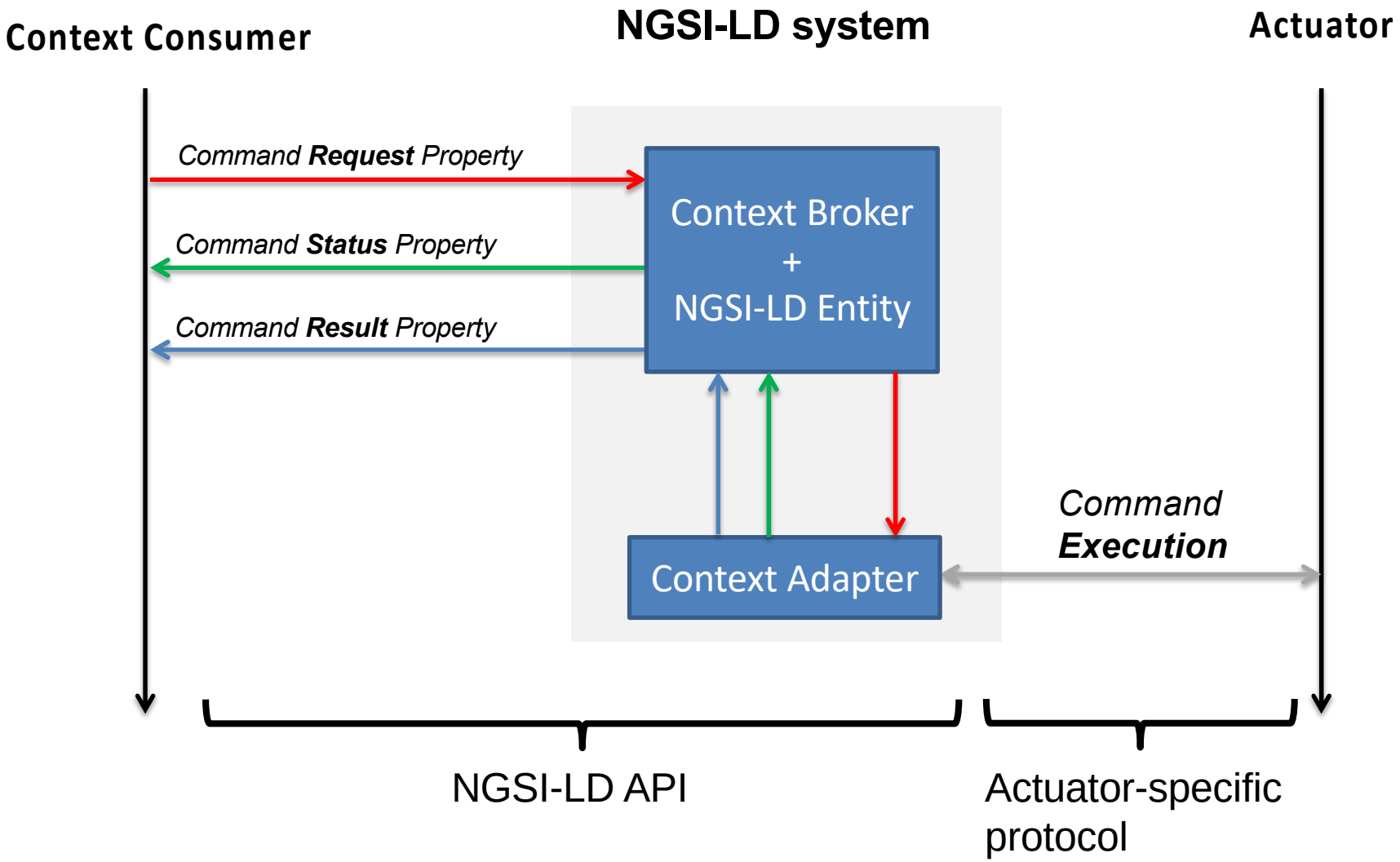
[CIM 009v1.6.1] **Clause 5.2.12**

Using NGSI-LD for Actuation (1)

- Regular NGSI-LD operations can be used for implementing actuations using **conventions** – important: the implementing Broker is not aware that the requests represent an ***actuation***
- For actuation, the **typical roles are inverted**, i.e. clients act as Context Producers that update properties to trigger actuations and actuation components (aka Context Adapters) act as Context Consumers
- **Actuation is triggered by updating a *command* property**, the actuation component provides *status* and *result* properties managed by the Broker (see next slide)

[CIM 009v1.6.1] **Annex H**

Using NGSI-LD for Actuation (2)



Using NGSI-LD for Actuation (3)
Conventions

- Available commands are provided in a *commands* property (e.g. “turn-on”)
- Updating the *command* property will trigger an actuation (under the assumption that an actuation component is actually “monitoring”!)
- By convention, there is a STATUS property, which has “-STATUS” concatenated to the name of the command property. The actuation component is responsible for updating it.
- By convention, there is a RESULT property, which has “-RESULT” concatenated to the name of the command property.
- Independently, there can be “regular” properties like “is-on” that are affected by the actuation
- “datasetId” can be used to distinguish multiple command instantiations, where supported

```
{
  "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": ...
  …
}
```

Using NGSI-LD for Actuation (4)
Implementation Options

- *Subscription / Notification* Model
  - Actuation components (aka Context Adapters) subscribe with the Broker for command properties. Whenever a client updates a command property, a notification with the new value is received and the actuation component can trigger the actuation. It would then update the status and result. Clients can subscribe to or query status and result.
- *Update Forwarding* Model
  - Actuation components register (as Context Sources) with the Broker for the entity / command properties using an exclusive registration.
  - As a result, the Broker forwards any update requests for the command properties. Status and results are handled as in the Subscription / Notification model.

Clarification on use of a default @context

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.

[CIM 009v1.6.1] **Clause 5.5.5**

Minor change to Scope representation

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`

The initial slash has been made optional. Example 1 and Example 2 are equivalent.

[CIM 009v1.6.1] **Clause 4.18**