New Features and Relevant Fixes in NGSI-LD v1.8.1

Martin Bauer

ETSI ISG CIM

12/04/2024

Overview (1)

- New Features in NGSI-LD v1.8.1
  - New Projection Parameters (pick, omit) – deprecation of attrs
  - Linked Entity Retrieval
  - datasetId parameter – enables “views” across attributes
  - New Attribute Types
    - 1:<n> relationships (unordered)
    - List relationships, 1:<n> (ordered)
    - List properties
    - JSON properties
  - Extended distributed queries
    - Id-only (creation of id-maps)
    - Loop-avoidance, using host alias (by tenant) and Via header, host alias retrievable via /info endpoint
  - Allow broader local requests
    - Queries(all) - not restricted to providing entity type(s) or attribute(s)
    - Subscriptions (all)
    - Linked Entity Retrieval (not requiring objectType)

[CIM 009v1.8.1] https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.08.01_60/gs_CIM009v010801p.pdf

Overview (2)

- Fixes and small features in NGSI-LD v1.8.1
  - `format` parameter, deprecating `options` parameter for specifying attribute representation
  - Relax restriction on forbidden characters
  - *Error code 504 instead of 503 for JSON-LD Context Access Points – clause 6.30.3.2*
  - Remove *scope* from attribute patch operation
  - *URI/terms clarification: both short-hand terms and URIs can be uses as property names and relationship names*
  - *URN namespace definitions: we interpret the full URN as case-insensitive, using lowercase*
  - GeoJSON type added as return type to Query Entities and Retrieve Entity figures
  - *Clarify match in distributed operations*
    added references to the respective clauses describing the matching to clauses 5.6.1, 5.6.2, 5.6.3, 5.6.4, 5.6.5, 5.6.6, 5.6.17, 5.6.18, 5.6.19
  - *Protect core context – Annex B: added line* `"@protected": true`
    As a result, terms from the core context cannot be overwritten by embedded @contexts. Such embedded @contexts are anyway only allowed at the top-level in NGSI-LD, not nested in inner nodes.
  - Fix in detailed description of hosted @contexts

[CIM 009v1.8.1] https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.08.01_60/gs_CIM009v010801p.pdf

New Projection Parameters (1)

- Motivation
  - The *attrs* parameter in NGSI-LD queries / attributes in NGSI-LD subscriptions has been used for **projection**, i.e. only returning the selected Attributes, but **at the same time also for selection**, i.e. only Entities having at least one of these Attributes are considered; Entities without any of these Attributes are discarded and there is no way to get "empty" entities (without Attributes) back.
  - The *q* parameter can also be used for **selection**, i.e. the existence of an attribute can be required, so there are practically two ways for selection
  - To separate the aspects of **projection** and **selection**, **two new projection parameters** are introduced that do not select at the same time: *pick* and *omit*; q can then be used for selection

[CIM 009v1.8.1] Clauses 4.21, **5.2.14**, **5.2.23**, 5.7.1, 5.7.2, 5.7.3, 5.7.4, 5.8.6, 6.4.3.2, 6.5.3.1, 6.18.3.2, 6.19.3.1

New Projection Parameters (2)

- *pick*: attribute name(s), and in addition "id", "type“ and "scope” elements to be returned in the result can be specified
- *omit*: attribute name(s), and in addition "id", "type“ and "scope” elements not to be returned in the result can be specified, i.e. all others, so this is the inverse of pick.
- [when omitting/not picking "id“ and/or "type“, the result will not contain valid NGSI-LD Entities anymore, i.e. cannot be used as the basis for subsequent NGSI-LD operations]
- The two parameters are mutually exclusive, i.e. they cannot both be used in the same request

```
GET /ngsi-ld/v1/entities/?type=WeatherForecast&format=simplified&pick=id,
type,humidity
Accept: application/json
Link: <http://example.org/myContext.jsonld>;
rel="http://www.w3.org/ns/json-ld#context";
type="application/ld+json"
```

```
[{   "id": "urn:ngsi-ld:WeatherForecast:XXX ",
     "type": “WeatherForecast",
     "humidity": 98 },
 {   "id": "urn:ngsi-ld:WeatherForecast:YYY",
     "type": “WeatherForecast"}
]
```

```
GET /ngsi-ld/v1/entities/?type=WeatherForecast&format=simplified&pick=id,
type,humidity&q=humidity
Accept: application/json
Link: <http://example.org/myContext.jsonld>;
rel="http://www.w3.org/ns/json-ld#context";
type="application/ld+json"
```

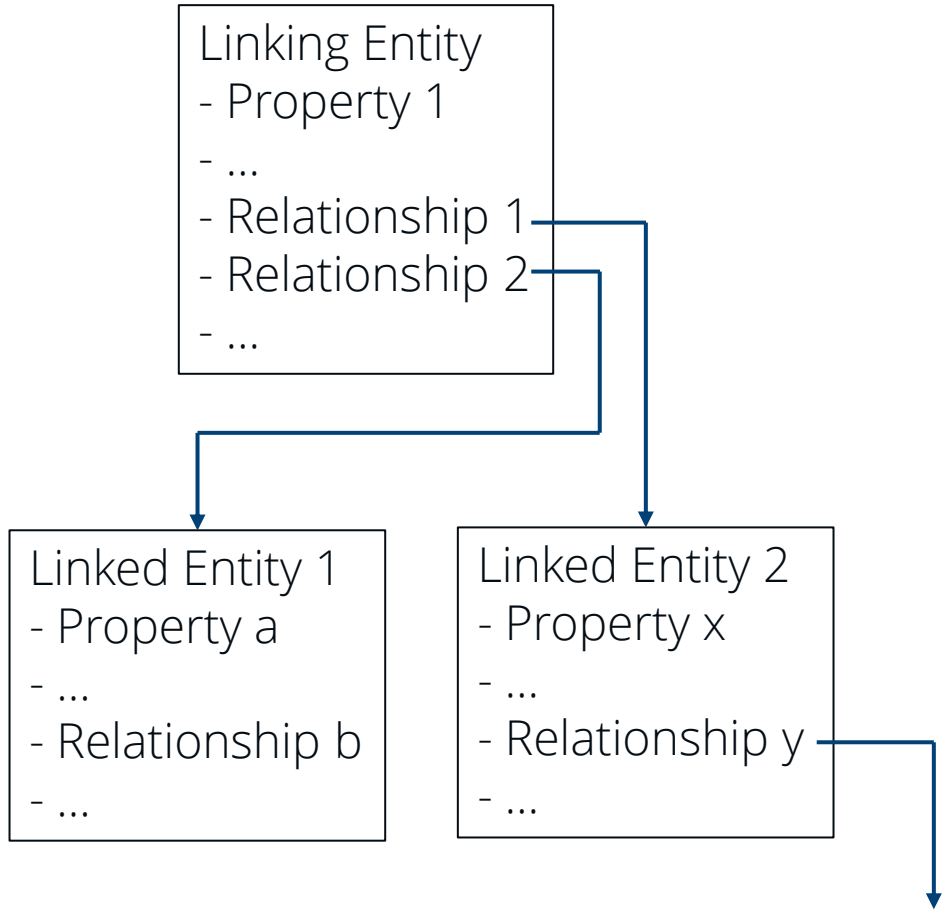
```
[{   "id": "urn:ngsi-ld:WeatherForecast:XXX ",
     "type": “WeatherForecast",
     "humidity": 98 }
]
```

`pick=id,type,humidity&q=humidity` **is equivalent to** `attrs=humidity`

Linked Entity Retrieval (1)

Since Entities are uniquely identifiable by a URI, it is possible to **traverse across the Entity graph** directly **from a Linking Entity to a Linked Entity**. It is therefore sometimes convenient to be able to **query or retrieve data via a single Context Broker request and to receive a response including both Linking Entities and dependent Linked Entities** directly.

The concept of Entity graph retrieval is a common concept amongst graph databases and it allows for more structured queries (see clause 4.9) and the complete serialization of an Entity and its dependents.



[CIM 009v1.8.1] Clauses 4.5.2.2, 4.5.2.3, 4.5.3.2, 4.5.3.3, 4.5.4, 4.5.22.2, 4.5.22.3, **4.5.23**, **4.9**, **4.21**, 5.2.6, 5.2.14.1, 5.2.23, 5.2.37, 5.7.1.3, 5.7.1.4, 5.7.1.5, 5.7.2.3, 5.7.2.4, 5.7.2.5, 6.4.3.2, 6.5.3.1, Annex B , Annex C

Linked Entity Retrieval (2) – Use Case

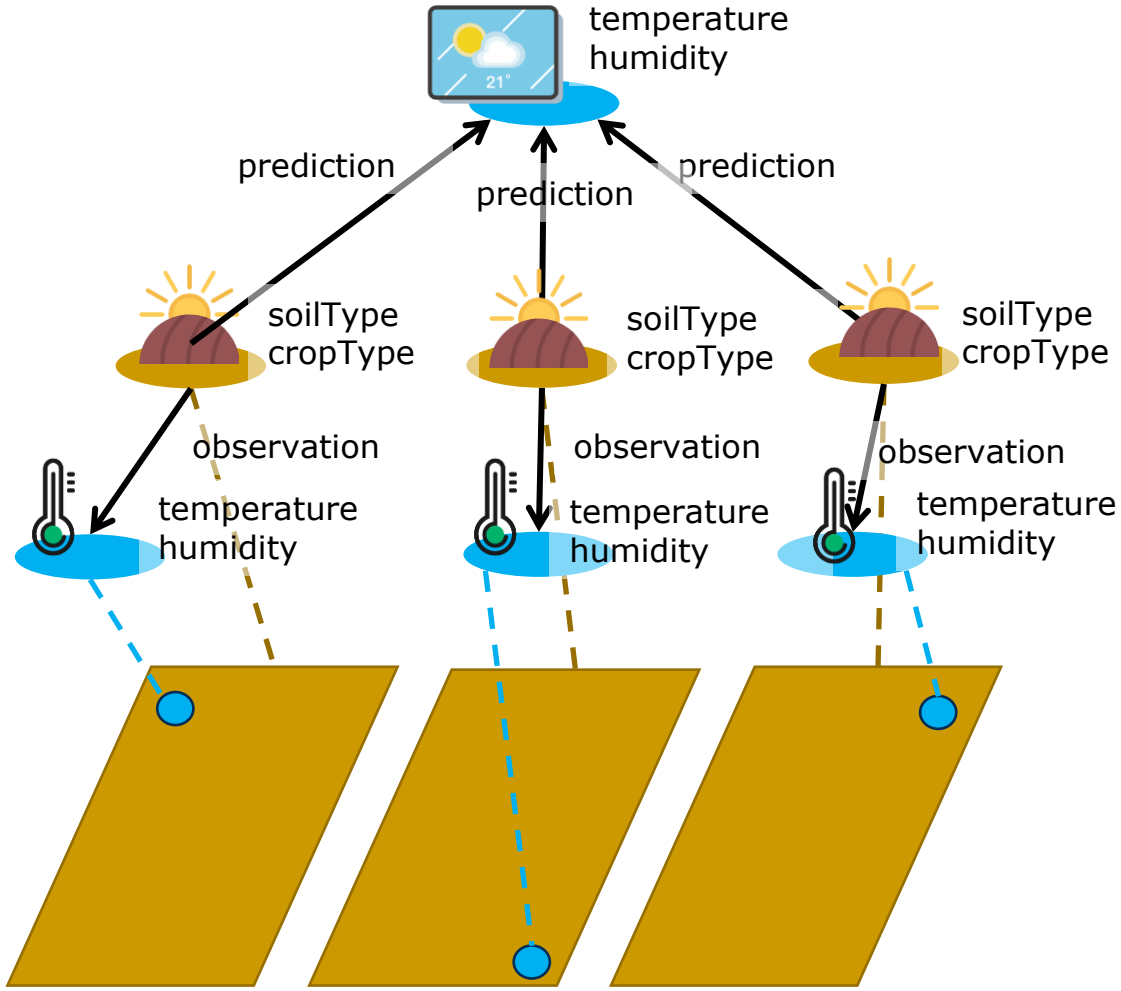
I hold several pieces of land **`type:AgriParcel`**

Each AgriParcel has some of its own attributes, e.g. **`soilType, cropType`**

Each AgriParcel also has its own related weather station (i.e. Relationship to an **entity** **`type:WeatherObserved`**) displaying values holding to current observed weather conditions

We also require a weather forecast (i.e. Relationship to an **entity** **`type:WeatherForecast`**) which covers a wider area encompassing multiple AgriParcels for a predicted weather conditions.

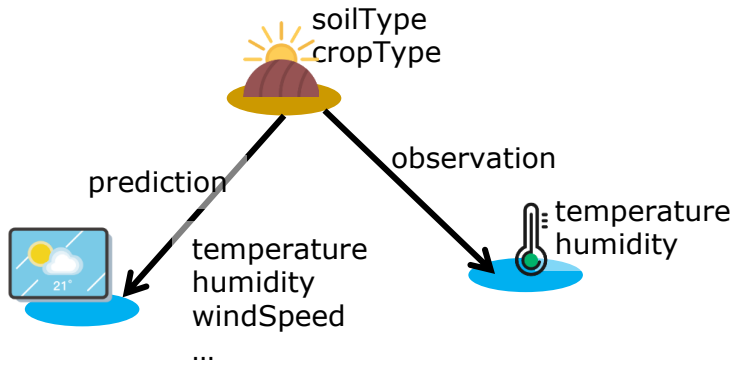
Note: **`WeatherObserved`** and **`WeatherForecast`** share similar attributes



7

Linked Entity Retrieval (3)

- NGSI-LD models graph of Entities through explicit *relationships* (e.g. *prediction* / *observation*)
- Up to now, retrieval can only be done step-by-step
- From v1.8 retrieval can be done for a whole subgraph (default: expand one level) – objectType required, unless *local* is specified in request




```
{
  "id": "urn:ngsi-ld:AgriParcel:001",
  "type": "AgriParcel",
  "soilType": "Loamy",
  "prediction": {
     "type": "Relationship",
     "object": "urn:ngsi-ld:WeatherForecast:XXX",
     "objectType": "WeatherForecast"
     },
  "observation": {
     "type": "Relationship",
     "object": "urn:ngsi-ld:WeatherObserved:001",
     "objectType": "WeatherObserved"
     }
}
```

Normalized Representation

Linked Entity Retrieval (4)

- With “join=inline”, they can be represented in a hierarchical structure

```
{
  "id": "urn:ngsi-ld:AgriParcel:001",
  "type": “AgriParcel",
  "soilType": "Loamy",
  "prediction": {
     "object": "urn:ngsi-ld:WeatherForecast:XXX",
     "objectType": “WeatherForecast",
     "entity": {
          "id": "urn:ngsi-ld:WeatherForecast:XXX ",
          "type": “WeatherForecast",
          "humidity": {"value": 98, "unitCode: "PCT"},
          "temperature": {"value": 30, "unitCode: "CEL"},
          "windSpeed": {"value": 3, "unitCode: "MPH"}
     }},
  “observation": {
     "object": "urn:ngsi-ld:WeatherObserved:001",
     "objectType": “WeatherObserved",
     "entity": {
          "id": "urn:ngsi-ld:WeatherObserved:001",
          "type": “WeatherObserved",
          …
}
```

Concise Representation

- With “join=flat”, the relationships stay as they are, but the target entities are returned in the result lists

```
[{
  "id": "urn:ngsi-ld:AgriParcel:001",
  "type": “AgriParcel",
  "soilType": "Loamy",
  "prediction": {
     "object": "urn:ngsi-ld:WeatherForecast:XXX",
     "objectType": “WeatherForecast }
  “observation": {
     "object": "urn:ngsi-ld:WeatherObserved:001",
     "objectType": “WeatherObserved",
 },
 {
  "id": "urn:ngsi-ld:WeatherForecast:XXX ",
  "type": “WeatherForecast",
  "humidity": {"value": 98, "unitCode: "PCT"},
  "temperature": {"value": 30, "unitCode: "CEL"},
  "windSpeed": {"value": 3, "unitCode: "MPH"}
 },
 {
  "id": "urn:ngsi-ld:WeatherObserved:001",
  "type": “WeatherObserved“}
 …
]
```

Concise Representation

Linked Entity Retrieval (5)

- Linked Entity Retrieval applies only to
  - Retrieve Entity (5.7.1, 6.5.3.1)
  - Query Entities (5.7.2, 6.4.3.2 / 6.23)
- Only Relationships of Entities are being followed, not meta information like Relationships of Properties or Relationships of Relationships.
- The following parameters are defined:
  - join ("inline" or "flat", as described on the previous slide)
  - joinLevel – Depth of Linked Entity retrieval, default is 1 (only if join is present)
  - containedBy - List of Entity Ids, which have previously been encountered whilst retrieving the Entity Graph. Avoids loops. Only applicable if joinLevel is present

Linked Entity Retrieval (5) – q Extension

- The filter can also apply to a Property or Relationship of an NGSI-LD Entity targeted by a (recursively) followed Relationship
- `?type=WeatherStation&join=flat&q=sensor{humidity}==40`

```
{
  "id": "urn:ngsi-ld:WeatherStation:123",
  "type": "WeatherStation",
  "sensor": {
    "type": "Relationship",
           "objectType": "Device",
    "object": "urn:ngsi-ld:Device:345"
  }
}
{
  "id": "urn:ngsi-ld:Device:345",
  "type": "Device",
  "humidity": {
    "type": "Property",
    "value": 40
  }
}
```

- As not knowing the Entity Type targeted by a Relationship could make the query significantly more expensive, a hint for the required Entity Type (e.g. Device) can be provided, so only such NGSI-LD Entities need to be considered.
- `?type=WeatherStation&join=flat&q=sensor{Device:humidity}==40`

Linked Entity Retrieval (6) – projection Extension

- With pick and omit, the projection is also extended to the Linked Entities, i.e., it can be specified, which Attributes are to be returned for the Linked Entities. This applies recursively up to the joinLevel.
- Examples:
  - `?pick=observation{temperature,humidity}`
  - `?pick=observation{temperature,humidity},prediction{temperature}`
  - `?omit=observation{device{windspeed}}`

datasetId parameter (1)

So far, **filtering** was **only** possible **on Entities with respect to the existence of an Attribute with a datasetId**, but you **could not return only the attribute instance with a certain datasetId**.

```
GET /ngsi-ld/v1/entities/?type=WeatherForecast&
q=temperature.datasetId="urn:ngsi-ld:WeatherForecast:datasetId:12345“
```

returns all Entities with a temperature Property, whose datasetId has the given value
(Entity with all its Attributes, in particular with all temperature Property instances)

- **datasetId is used as a new parameter for consumption and subscription operations.** It specifies a comma-separated list of one or more datasetIds – only attribute instances that have one of the specified datasetIds are returned.
- **@none** is used for specifying the **default datasetId** (i.e. matching default attribute instances without a specified datasetId)

[CIM 009v1.8.1] Clauses **4.5.5**, 5.2.9, 5.7.1, 5.7.2, 5.7.3, 5.7.4, 5.12, 6.4.3.2, 6.5.3.1, 6.18.3.2, 6.19.3.1

datasetId parameter (2) – enables “views” across attributes

- If the **same datasetId is used across multiple attributes**, a ***view*** can be implemented, e.g. using “urn:12345” and “urn:98765” for two different views.

```
{
     "id": "urn:ngsi-ld:WeatherForecast:XXX ",
     "type": “WeatherForecast",
     "humidity": [{"type": "Property", "value": 98, "unitCode: "PCT“, datasetId=“urn:12345"},
          {"type": "Property", "value": 98, "unitCode: "PCT“, datasetId=“urn:98765"}],
     "temperature": [{"type": "Property", "value": 30, "unitCode: "CEL", datasetId=“urn:12345"},
          {"type": "Property", "value": 86, "unitCode: “FAH", datasetId=“urn:98765"}],
     "windSpeed": [{type": "Property", "value":5, "unitCode: “KMH“, datasetId=“urn:12345"},
          {type": "Property", "value":3, "unitCode: “MPH“, datasetId=“urn:98765"}]
 }
```

```
GET /ngsi-ld/v1/entities/?type=WeatherForecast&datesetId=urn:12345
Accept: application/json
Link: <http://example.org/myContext.jsonld>; rel="http://www.w3.org/ns/json-ld#context";
type="application/ld+json"
```

```
[{
     "id": "urn:ngsi-ld:WeatherForecast:XXX ",
     "type": “WeatherForecast",
     "humidity": {"type": "Property", "value": 98, "unitCode: "PCT“, datasetId=“urn:12345"},
     "temperature": {"type": "Property", "value": 30, "unitCode: "CEL", datasetId=“urn:12345"},
     "windSpeed": {type": "Property", "value":5, "unitCode: “KMH“, datasetId=“urn:12345"}
 }]
```

datasetId parameter (3)

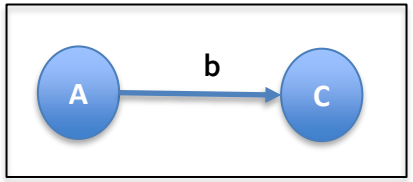
- **datasetId** is also a **parameter in CSourceRegistrations**
- Its value is a list of datasetIds that indicates for which datasetIds the registered Context Source may have Attribute instances
- Again, this can be useful when using the datasetId for implementing views

New Attribute Types - Overview

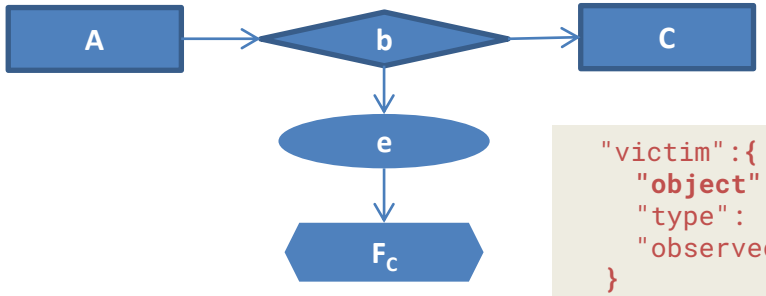
- 1:<n> relationships (unordered)
  - This is really an extension of the Relationship type, allowing multiple target Entities as an array, which is the value of the “object”. JSON-LD does not guarantee a fixed order.
- List relationships, 1:<n> (ordered)
  - The ListRelationship type has an “objectList” instead of an “object”, which is always an ordered JSON-LD list of Entity Ids that can have any number of elements (1 to <n>)
- List properties (ordered)
  - The ListProperty type has a “valueList” instead of a “value”, which is always an ordered JSON-LD list of JSON values.
- JSON properties
  - The JsonProperty type has a “json” instead of a “value”, which contains json data that is “ignored” by JSON-LD, i.e. it is not subject to JSON-LD expansion and compaction, and thus is kept exactly “as-is”.

1:<n> relationships (unordered) (1)

- **Relationships** (where metadata (F_C) can be attached, property-graph model)

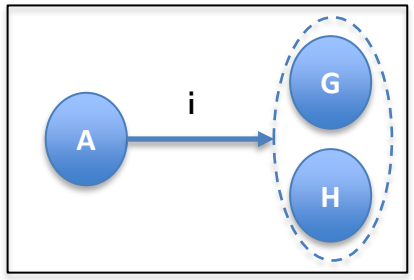


high-level graph view

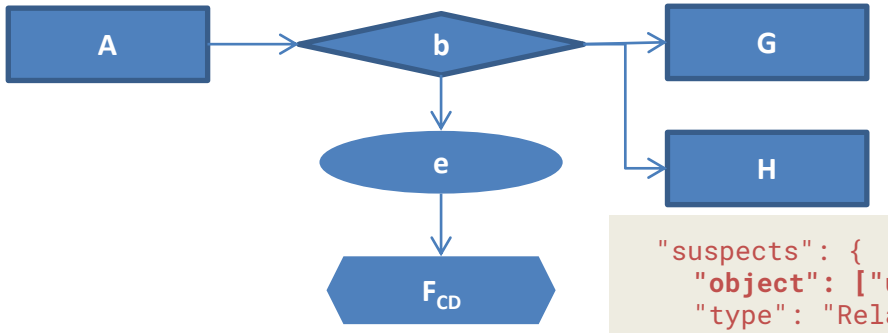



```
"victim":{
  "object": "urn:sam-rachet",
  "type": "Relationship",
  "observedAt": "01-01-2023"
}
```

- **New: 1-n Relationships - unordered** (one Entity is related to a set of n Entities, with a single meta information that can only be updated together) – object can not only be a single URI, but a list of URIs.



high-level graph view




```
"suspects": {
  "object": ["urn:miss-scarlett","urn:rev-green],
  "type": "Relationship",
  "observedAt": "01-01-2023"
}
```

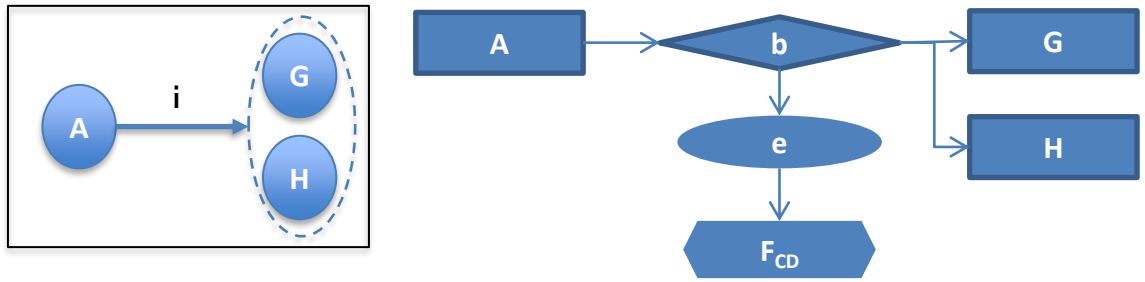
[CIM 009v1.8.1] **Clauses 4.5.3.2**, 4,5.4

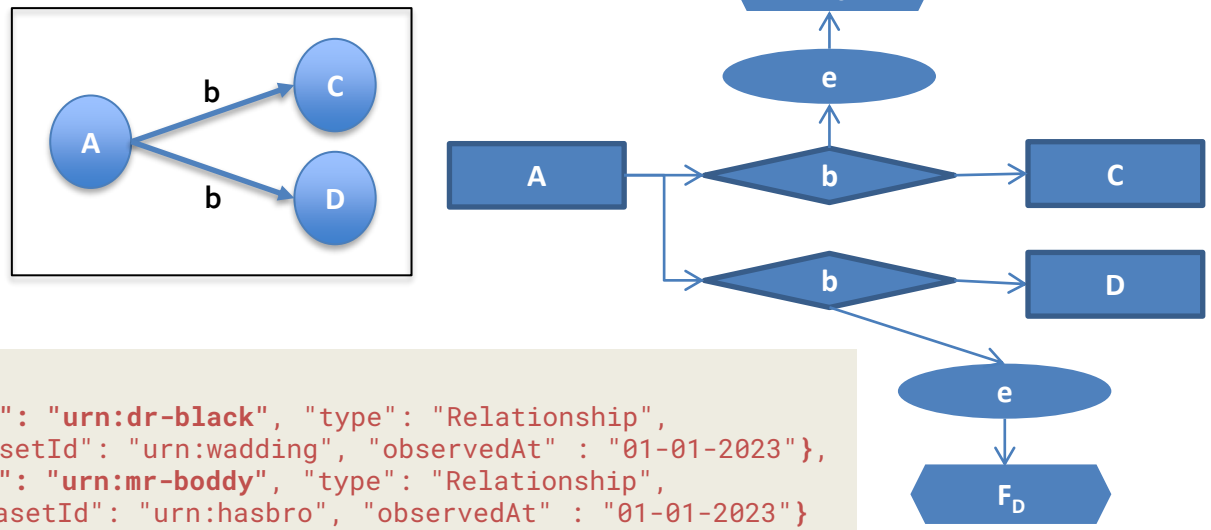
1:<n> relationships (unordered) (2)

Structural comparison 1:<n> relationships vs. multi-relationships

1-n Relationships - unordered (one Entity is related to a set of n Entities, with a single meta information, can only be updated together)

Multi-Relationship (one Entity has the same type of relationship with (individual or set) of other entities. Each such relationship has separate metadata (FC and FD) and can be independently updated using datasetId as identifier, except for at most one default without datasetId.)






```
"suspects": {
   "object": ["urn:miss-scarlett","urn:rev-green],
   "type": "Relationship“,
   "observedAt": "01-01-2023"
}
```

```
"victim": [
  { "object": "urn:dr-black", "type": "Relationship",
      "datasetId": "urn:wadding", "observedAt" : "01-01-2023"},
  { "object": "urn:mr-boddy", "type": "Relationship",
      "datasetId": "urn:hasbro", "observedAt" : "01-01-2023"}
]
```

List relationships, 1:<n> (ordered)

For the ordered 1:<n> relationship, a new Relationship type `ListRelationship` has been introduced to use the `@list` definition for the JSON-LD element `objectList` in the NGSI-LD core context that ensures the order.

Normalized - an ordered 1-N relationship

```
"suspects": {
  "objectList": ["urn:miss-scarlett","urn:rev-green","urn:col-mustard",
                 "urn:prof-plum", "urn:mrs-peacock","urn:mrs-white"],
  "type": "ListRelationship“,
  "observedAt": "01-01-2023"
}
```

Concise - presence of **objectList** is sufficient to indicate an ordered 1-N relationship

```
"suspects": {
  "objectList": ["urn:miss-scarlett","urn:rev-green","urn:col-mustard",
                 "urn:prof-plum", "urn:mrs-peacock","urn:mrs-white"],
  "observedAt": "01-01-2023"
}
```

Simplified - list of URNs is just returned as an Array

```
"suspects": ["urn:miss-scarlett","urn:rev-green","urn:col-mustard",
             "urn:prof-plum", "urn:mrs-peacock","urn:mrs-white"]
```

[CIM 009v1.8.1] Clauses 4.5.4, 4.5.9, **4.5.22**, 5.2.4, 5.2.5, 5.2.6, 5.2.7, 5.2.31, 5.2.32, 5.2.35, 5.2.36, **5.2.37**, 5.2.38

List properties (ordered)

Corresponding to the `ListRelationship` type, a `ListProperty` type has been introduced that guarantees the order with the JSON-LD element `valueList` defined as `@list` in the NGSI-LD core context.

Normalized - an ordered 1-N array of JSON objects or primitives

```
"numbers": {
   "valueList": [10, 12, 14, 16 18],
   "type": “ListProperty"
}
```

Concise - presence of **`valueList`** is sufficient to indicate an ordered 1-N array

```
"numbers": {
   "valueList": [10, 12, 14, 16 18]
}
```

Simplified - list of URNs is returned as an Array

```
"numbers": [10, 12, 14, 16 18]
```

[CIM 009v1.8.1] Clauses 4.5.4, 4.5.9, **4.5.21**, 5.2.4, 5.2.5, 5.2.6, 5.2.7, 5.2.31, 5.2.32, 5.2.35, **5.2.36**, 5.2.37, 5.2.38,

JSON Property (1)

The value of a JSON Property is not expanded by the JSON-LD @context.

To enable this behaviour, the JSON-LD key has to be defined as @json in the JSON-LD context. Thus, a JSON Property has a “json”, instead of the “value” of a regular Property.

```
"json": {
      "@id": "ngsi-ld:hasJSON",
      "@type": "@json"
    }
```

Example:

```
"parkingTickets": {
  "type": "JsonProperty",
  "json": {
     "id": "85a6cc52-0589-45f9",
     "value": "Overstay 60
minutes"
  }
}
```

Expanded with suitable @context

```
[
  {
    "https://uri.etsi.org/ngsi-ld/default-context/hasParkingTickets": [
      {
        "https://uri.etsi.org/ngsi-ld/hasJSON": [
          {
            "@type": "@json",
            "@value": {
              "id": "85a6cc52-0589-45f9",
              "value": "Overstay 60 minutes"
            }
          }
        ],
        "@type": [
          "https://uri.etsi.org/ngsi-ld/JsonProperty"
        ]
      }
    ]
  }
]
```

unexpanded

[CIM 009v1.8.1] Clauses 3.1, 4.5.2.2, 4.5.2.3, 4.5.3.2, 4.5.3.3, **4.5.4**, **4.5.9**, **4.5.24**, **4.9**, 5.2.2, 5.2.4, 5.2.5, 5.2.6, 5.2.7, 5.2.31, 5.2.32, 5.2.35, 5.2.36, 5.2.37, **5.2.38**, 5.5.7, **Annex B**, C.2.2

JSON Property (2)

- In order to filter according to JSON-Properties using the `q` parameter, the filtering component needs to know, which content elements in the filter are not to be expanded. For this purpose, a second parameter, `jsonKeys`, has been introduced that identifies the Attributes, whose values are not to be expanded using the applicable @context, so that the filter can properly be applied.
- EXAMPLE:

```
?q=parkingTickets[value]==”Overstay 60 minutes”&jsonKeys=parkingTickets.
{
  "id": "urn:ngsi-ld:Car:6152s",
  "type": "Car",
  "parkingTickets": {
            "type": "JsonProperty",
            "json": {
                      "id": "85a6cc52-0589-45f9",
                      "value": "Overstay 60 minutes"
            }
  }
 }
```

- Without `parkingTickets` specified in jsonKeys, `value` in the `q` filter would be expanded to https://uri.etsi.org/ngsi-ld/hasValue (based on the NGSI-LD core context) and thus would not match the `value` element within the json value, which is not expanded due to the definition of `json`.

Extended Distributed Queries (1) – Entity Maps

- **To properly support queries for distributed Entities (pagination), the set of Entities has to be frozen** and the information collected. For this purpose, Entity Maps have been introduced, so subsequent requests can use the same set of Entities.

NGSI-LD Entity Map: A mapping of NGSI-LD Entity ids to Context Source Registrations used in maintaining atomicity of transactions performed by Distribution Brokers and Federation Brokers

- In distributed operations, the creation of an Entity Map can be requested, whose location is then returned
- In subsequent related requests (paging), the location of the resource holding the Entity Map is provided, so only the entities and related registrations available at the time of the original request are considered.

[CIM 009v1.8.1] Clauses 3.1, **4.3.6.7**, **4.5.25**, **5.2.39**, **5.5.14,** 5.7.1, 5.7.2, 5.12, **5.14**, 6.2, 6.4.3.2, 6.5.3.1, **6.32**

Extended Distributed Queries (2) – Entity Maps

Example of an EntityMap

```
{
  "id": "urn:ngsi-ld:entitymap1234",
  "type": “EntityMap",
  "expiresAt": "2024-03-12T12:05:02Z",
  "entityMap": {
     "urn:ngsi-ld:AgriParcel:001": ["@none“,
"urn:ngsi-ld:ContextSourceRegistration:csr1a3456"],
     "urn:ngsi-ld:AgriParcel:002": ["@none“,
"urn:ngsi-ld:ContextSourceRegistration:csr2b8765"]
   },
  "linkedMaps": {
          "urn:ngsi-
ld:ContextSourceRegistration:csr1a3456": "urn:ngsi-
ld:EntityMap:em1c23456",
          "urn:ngsi-
ld:ContextSourceRegistration:csr2b8765": "urn:ngsi-
ld:EntityMap:em2e098763"
   }
}
```

Interface for retrieving and managing Entity Maps

Retrieve	/entityMaps/{entityMapId}	GET	Entity Map Retrieval by Id
Update	/entityMaps/{entityMapId}	PATCH	Entity Map Update by Id *
Delete	/entityMaps/{entityMapId}	DELETE	Entity Map Deletion by Id

*** only `expiresAt` can be updated**

- `entityMap` contains a set of key-value pairs whose keys shall be strings representing Entity ids and whose values shall be an array holding every CSourceRegistration id which is relevant to the ongoing Context Information Consumption request.
  The key `"@none"` shall be used to refer to an Entity that is held locally.
- `linkedMaps` contains a set of key-value pairs whose keys shall be strings representing CSourceRegistration ids which are relevant to the ongoing Context Information request and whose values shall represent the associated EntityMap id used by the ContextSource.

Extended Distributed Queries (3) – Loop Avoidance

Avoid Loops in Distributed Requests

- **General approach: It is not known if any distributed endpoints of a registered Context Source are in turn reliant on previously encountered context sources thus causing an infinite loop.** Therefore, when processing a distributed operation, **a specific field listing all previously encountered Context Sources shall be passed as part of the request** and this field can be used to exclude duplicated sources from matching as context source registrations.
- **In HTTP binding: use of Via header** (IETF RFC 7230 [27]))
- Requires: compatible **hostAlias, per Tenant**, used in CSourceRegistrations
- New **endpoint for retrieving hostAlias**, based on tenant
  - GET /ngsi-ld/v1/info/sourceIdentity, using Tenant header, if not for default tenant

[CIM 009v1.8.1] Clauses **4.3.6.4**, **5.2.40**, 5.12, **5.15**, **6.3.18**, **6.33**

Allow broader local requests

- Allow broader requests in the local case, e.g.
  - Specifying just the Entity IDs.
  - Not specifying anything, i.e. querying / subscribing to all information
- The reason that this is not generally allowed is that in distributed cases this would always require contacting ALL Brokers and Context Sources in the system for all information they have, which would be too expensive.
- Queries and Subscriptions
  - For queries: specify local=true in the request, default is local=false (setting type to "*" is alternatively allowed)
  - For subscription: set the localOnly member in the subscription (setting type to "*" in the EntitySelector of the Subscription is alternatively allowed)
- Linked Entity Retrieval (not requiring objectType) [see slide "LinkedEntityRetrieval"]

[CIM 009v1.8.1] Clauses 5.2.12, 5.2.33, **5.5.13**, 5.7.2, 5.7.3, 5.7.4, 6.3.18, 6.4.3.2, 6.18.3.2, 6.19.3.1

format Parameter (1) – representation of Entities

The use of the `options` parameter for specifying attribute representation is **deprecated**. The goal is to avoid that the same parameter is used for different purposes. The preferred option for specifying the attribute representation now is the `format` parameter. All other uses of the `options` parameter are not affected.

For operations regarding regular Entities, the following `format` values can be used:

- `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.
- `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 Context Broker will return data in the most concise lossless representation possible, for example removing all Attribute `type` members.
- `simplified` (or its synonym `keyValues`): 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.

[CIM 009v1.8.1] Clauses **6.3.7**, **6.3.12**, 6.5.3.4,

format Parameter (2) – representation of Temporal Entities

Deprecating `options` parameter for specifying attribute representation. The goal is to avoid that the same parameter is used for different purposes. The preferred option for specifying the attribute representation is not the `format` parameter. All other uses of the `options` parameter are not affected for now.

For operations regarding regular Temporal Entities, the following `format` values can be used:

- `temporalValues:` a simplified temporal representation of entities shall be provided as defined by clause 4.5.6.
- `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. If both `format` and `options` are present, the value of the format parameter shall take precedence.

[CIM 009v1.8.1] Clauses 5.2.14, 5.8.6, **6.3.7**, **6.3.12**, 6.5.3.4, C.5.3.2, C.5.4.2, C.5.4.3, C.5.6.2, C.5.13.2, G.3.1

Relax restriction on forbidden characters

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, **implementations should consider that the incoming data from a client may contain the following characters**:

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

When receiving entities (context information) encoded in JSON format and containing values that include the above characters, **implementations should decide how to resolve the possible security problems that may be generated by the data**. In all cases, **implementations shall preserve the representation of the content of the values** provided by the context information providers and return the original content when replying to context consumption requests.
If implementations decide to raise an error, the error shall be BadRequestData.

[CIM 009v1.8.1] Clause **4.6.4**

Remove *scope* from attribute patch operation

If the target Attribute is *scope*, then an error of type ***BadRequestData*** shall be raised.

Reason: scope is not reified, so the content is just a String or array of Strings. This is not a JSON-LD document, but on the other hand the content is required to be a JSON-LD document.

Scope can still be patched with an Entity patch, which is not much more verbose.

[CIM 009v1.8.1] Clause **5.6.4**

GeoJSON type added as return type to Query Entities and Retrieve Entity figures

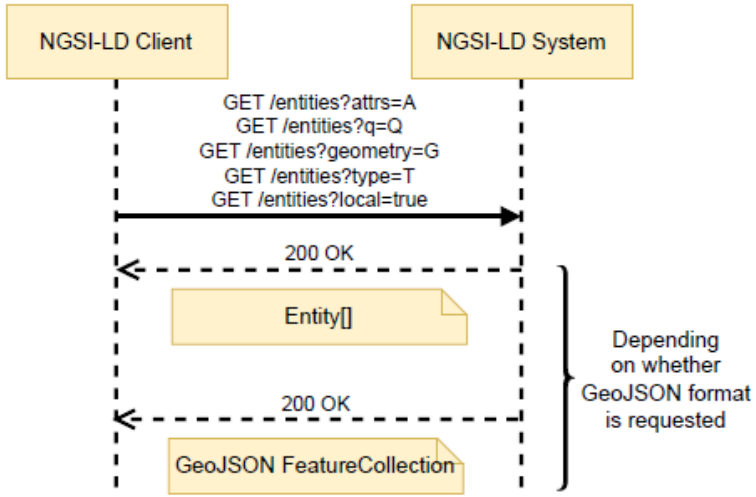


Figure 6.4.3.2-1: Query Entities interaction

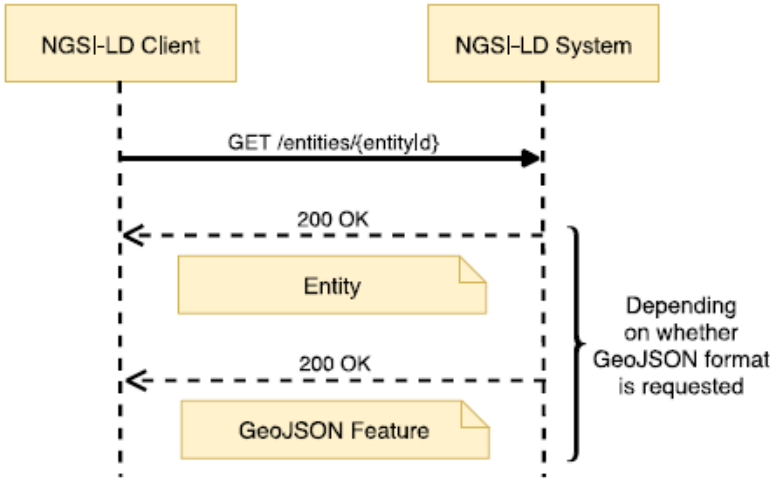


Figure 6.5.3.1-1: Retrieve Entity interaction

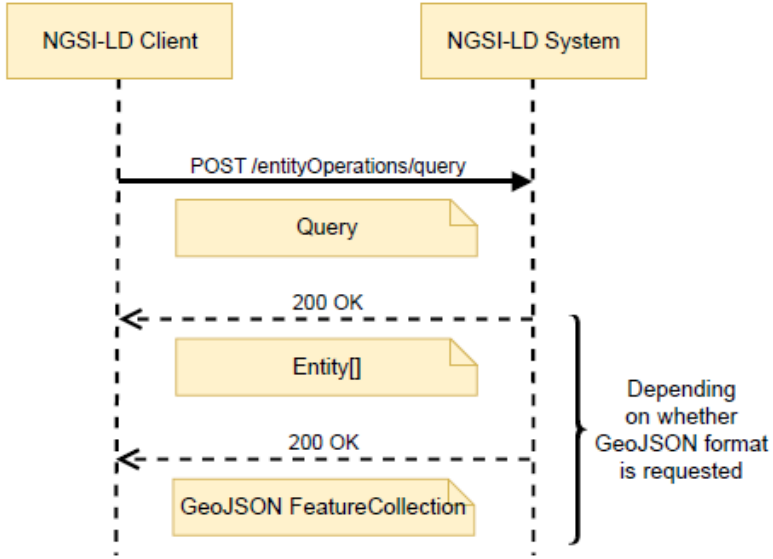


Figure 6.23.3.1-1: Query Entity via POST Interaction

[CIM 009v1.8.1] Clauses 6.4.3.2, 6.5.3.1, 6.23.3.1