New Features and Changes in NGSI-LD v1.3.1

Presented by: **Martin Bauer**

For: **ETSI-ISG CIM**

14.07.2020

Overview

Incompatible Changes compared to NGSI-LD API v1.2.1

- Important: Changes to Core @context / Migration options
- Multi-Attribute Support
- Query Language Syntax Changes to Attribute Path
- Batch Operation Error Codes

New Features in NGSI-LD API v1.3.1

- Query for available Entity Types and Attributes
- Full GeoJSON documents as query responses and notifications
- MQTT Notification Binding
- COUNT header option
- Multi-Tenant Support
- POST Queries
- Specification of additional headers for notifications
- Support for queries where attribute is implicitly specified in query filter or geoquery
- Support for international characters in URIs/IRIs and terms

Important:
Change: Core @context / Migration options

Core @context: in particular CIM009v1.3.1 Annex B

Incompatible Changes to NGSI-LD Core Context

- **Due to the support of retrieving results as GeoJSON / GeoJSON-LD documents (see slides 31-36), the NGSI-LD core context had to be changed in an incompatible way as the GeoJSON terms have to correspond to the GeoJSON-LD terms, which was not previously the case**
- In addition we did some additional minor changes to the core context (see following slides)
  - Alignment of temporal terms
  - Avoiding clashes with commonly used terms /reuse of widely used terms
- To avoid issues for existing v1.2.1 applications and Brokers, the core context URL changes from
  - https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld (v1.2.1) to
  - https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.3.jsonld (from v1.3.1)
- The different migration options are discussed on slide 8
  This change affects existing NGSI-LD data. In particular, the value of GeoProperties stored in a Broker uses a different URI, so when using the new @context there is a mismatch. This needs to be explicitly addressed.

Core @context incompatibilities (1)

v1.2.1	v1.3.3	Impacted Datatypes (and Operations)	Comment

"coordinates": "**ngsi-ld:**coordinates"	"coordinates": {
	"@container": "@list",
	"@id": "**geojson:**coordinates"
	}
		All using GeoProperties! (e.g. Create, Update, Append, Retrieve, Query, Notify)
		GeoQuery(Subscription)
			Uses GeoJSON namespace
			Uses JSON-LD 1.1 feature ("@container": "@list“), which preserves nested JSON arrays in JSON-LD, was a problem in JSON-LD 1.0)

"description": "**ngsi-ld**:description"	"description": "http://purl.org/dc/terms/description"	CSourceRegistration (optional), Subscription (optional)	Use term from Dublin Core as term is widely used

"end": {
"@id": "ngsi-ld:end",
"@type": "DateTime"
}
	"end**At** ": {
	"@id": "ngsi-ld:endAt",
	"@type": "DateTime"
	}
		TimeInterval (CSourceRegistration, temporal case)	Temporal, add “At” for consistency

"endTime": {
"@id": "ngsi-ld:endTime",
"@type": "DateTime"
}
	"endTime**At** ": {
	"@id": "ngsi-ld:endTimeAt",
	"@type": "DateTime"
	}
		TemporalQuery (also query parameter!)	Temporal, add “At” for consistency

"expires": {
"@id": "ngsi-ld:expires",
"@type": "DateTime"
}
	"expires**At** ": {
	"@id": "ngsi-ld:expiresAt",
	"@type": "DateTime"
	}
		CSourceRegistration, Subscription	Temporal, add “At” for consistency

"geometry": “**ngsi-ld**:geometry"	"geometry": "**geojson**:geometry"	GeoQuery(Subscription)	Uses GeoJSON namespace

Core @context incompatibilities (2)

v1.2.1	v1.3.3	Impacted Datatypes & Operations	Comment
"name": "ngsi-ld:name"		CSourceRegistration, Subscription	Replaced by "subscriptionName" and "registrationName" respectively
	"properties": "**geojson**:properties"	New: GeoJSON Representation	New as needed for GeoJSON – old ngsi-ld:properties has become ngsi-ld:propertyNames
"properties": { "@id": "ngsi-ld:properties", "@type": "@vocab" }	"propert**yNames** ": { "@id": "ngsi-ld:propertyNames", "@type": "@vocab" }	RegistrationInfo (CSourceRegistration)	Replacing "properties" as term is defined in GeoJSON
	"**registration**Name": "ngsi-ld:registrationName"	CSourceRegistration	Replaces "name" in registrations
"relationships": { "@id": "ngsi-ld:relationships", "@type": "@vocab" }	"relationship**Names**": { "@id": "ngsi-ld:relationshipNames", "@type": "@vocab" }	RegistrationInfo (CSourceRegistration)	Replacing "relationships" to align with propertyNames
"start": { "@id": "ngsi-ld:start", "@type": "DateTime" }	"start**At**": { "@id": "ngsi-ld:startAt", "@type": "DateTime" }	TimeInterval (CSourceRegistration, temporal case)	Temporal, add "At" for consistency

Core @context incompatibilities (3)

v1.2.1	v1.3.3	Impacted Datatypes & Operations	Comment
	"**subscription**Name": "ngsi-ld:subscriptionName",	Subscription	Replaces "name" in subscriptions

"time": {
"@id": "ngsi-ld:time",
"@type": "DateTime"
}
	"time**At**": {
	"@id": "ngsi-ld:timeAt",
	"@type": "DateTime"
	}
		TemporalQuery (also query parameter!)	Temporal, add "At" for consistency

"title": "**ngsi-ld**:title"	"title": "http://purl.org/dc/terms/title"	Error description	Use term from Dublin Core as term is widely used

What are the migration options for core @context?

ETSI ISG CIM can only describe the options, ultimately it is up to the implementers

- Smooth vs. hard transition
- Effort required
- General: Providing a script for updating existing data on migration recommended …

	Application with old core @context	Application with new core @context
Old Broker	✓	✗
New Broker – hard transition	✗ reject	✓
New Broker – replace with new assumption: existing data has been updated	(✓) – internally using new only, application has to deal with new @context in reply	✓
New Broker – core @context aware	✓ Support both externally, depending on which is provided → complex	✓

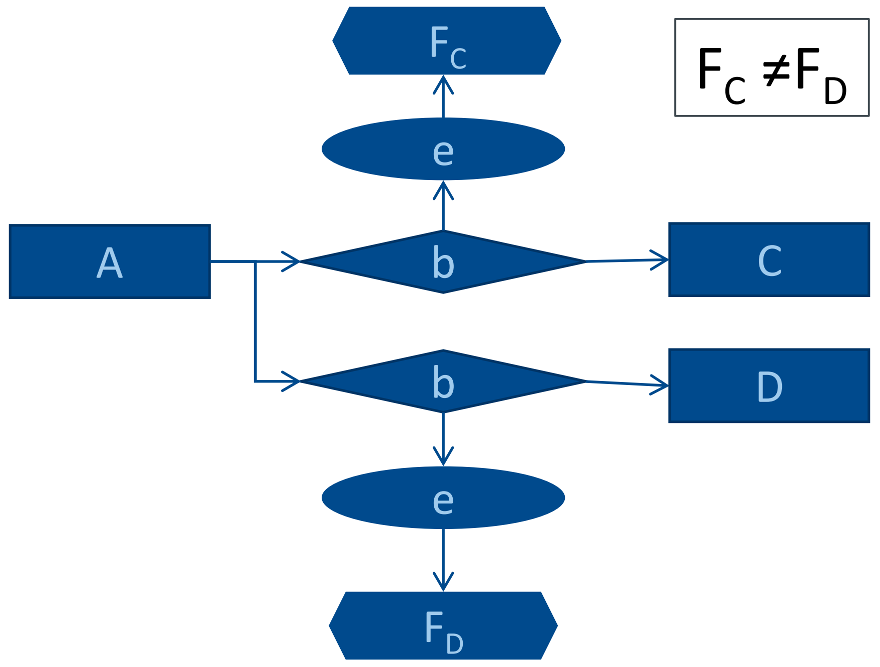
Change: Multi-Attribute Support

Multi-Attributes: in particular CIM009v1.3.1 Clauses 4.5.5 and 4.5.1

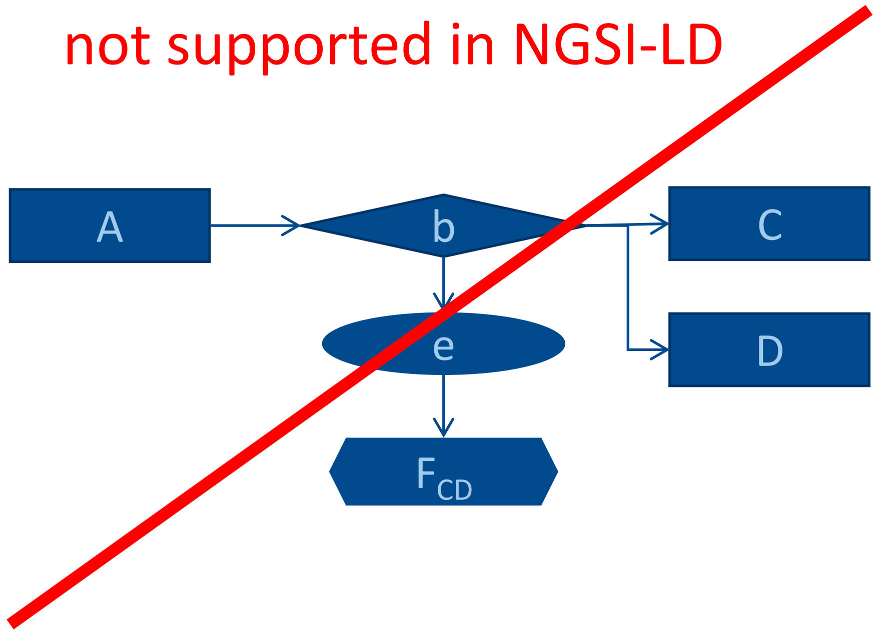
Multi-Attributes (Multi-Properties and Multi-Relationships)

- Multi-attributes means that multiple attribute instances of the same name exist.
- Typical use case are the following:
  - **Properties:** the same information is provided by different sources, e.g. for redundancy purposes, or there simply exist different sources that independently provide the same information. For example, the speed of a car may be provided by the speedometer, a GPS system or a stationary speed measurement system. These measurements may have different qualities and it should be possible to keep them independently as “speed” attribute instances of the same car. To differentiate the attributes from the different sources, a datasetId is used and each source updates the instance with its datasetid.
  - **Relationships:** Not all relationships are functional, i.e. an entity may have the same kind of relationship with different entities, e.g. “isFriendOf”, “hasChild”, etc. These relationships may have different properties, i.e. “isBestFriend=true”.
- Multi-attributes already existed in NGSI v1.2.1, but the representation was not straightforward.

Multi-Relationship Modelling



≠

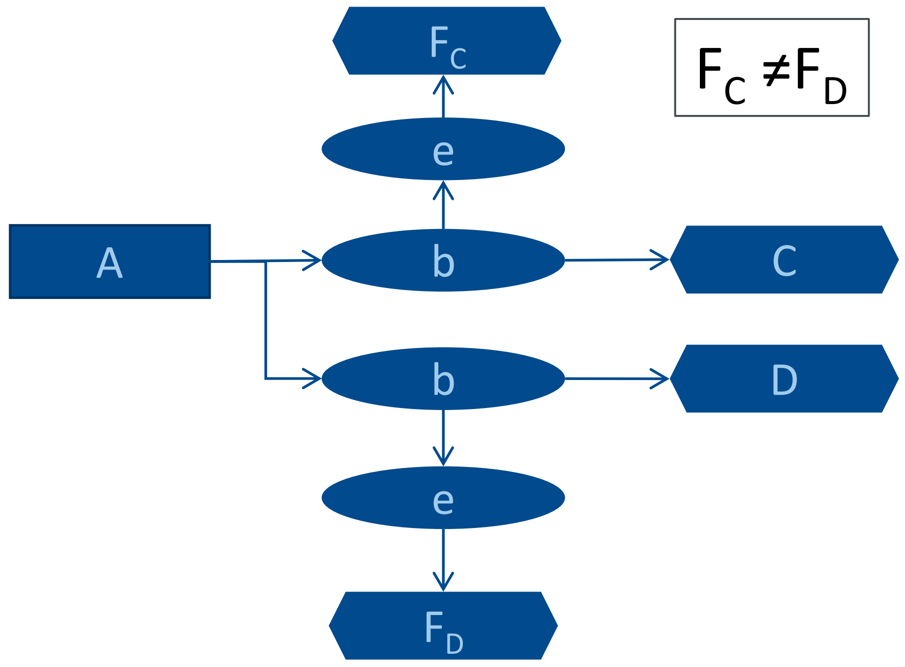


✓

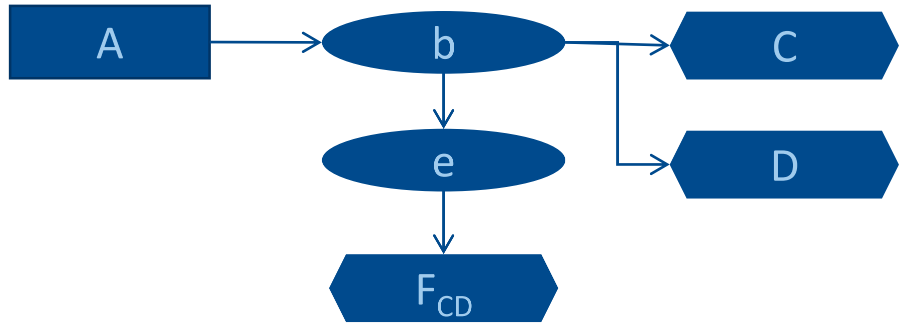
A has Relationships “b” with Entities C and D and each relationship “b” has a property “e”.

✗ Relationship with multi-object (JSON representation where there is an array of objects)

Multi-Property Modelling



≠



✓

A has two independent Properties “b” with values C and D respectively and each Property “b” has a property “e”.

✗ Property with multi-value (JSON representation where value is an array)

Current Simple Relationship Representation

```
"friend": {
    "type": "Relationship",
    "object": "urn:person:Sam",
    "friendInfo": {
        "type": "Property",
        "value": {
            "nickname": "Sammy",
            "age": 22,
            "description": "Best-friend"
        }
    }
}
```

Example: from SeungMyeong Jeong, KETI

Multi-Attribute Representation in NGSI v1.2.1

```
{
@context: {
"friend#1": "https://example.org/myFriend",
"friend#2": "https://example.org/myFriend",
     …
},
…
"friend#1": {
       "type": "Relationship",
       "object": "urn:person:Sam",
       "datasetId": "urn:myspecialfriendid1234567",
       "friendInfo": {
          "type": "Property",
          "value": {
             "nickname": "Sammy",
             "age": 22,
             "description": "Best-friend"
          }
       }
   },
"friend#2": {
       "type": "Relationship",
       "object": "urn:person:Victoria",
       "datasetId": "urn:myspecialfriendid9876543",
       "friendInfo": {
          "type": "Property",
          "value": {
             "nickname": "Vicky",
             "age": 25
          }
       }
   }
….
}
```

Multi-Attribute Representation in NGSI v1.3.1

```
"friend": [
  {
    "type": "Relationship",
    "object": "urn:person:Sam",
    "datasetId": "urn:myspecialfriendid1234567",
    "friendInfo": {
      "type": "Property",
      "value": {
        "nickname": "Sammy",
        "age": 22,
        "description": "Best-friend"
      }
    }
  },
  {
    "type": "Relationship",
    "object": "urn:person:Victoria",
    "datasetId": "urn:myspecialfriendid9876543",
    "friendInfo": {
      "type": "Property",
      "value": {
        "nickname": "Vicky",
        "age": 25
      }
    }
  }

]
```

Example: from SeungMyeong Jeong, KETI

Pros and Cons of the New Representation in v1.3.1

- Pros
  - No adapted @context needed
  - No hash-tagged elements
  - all relationships / properties under the same key
  - JSON-LD way of handling single vs. multi-value cases

  → greatly simplifies processing
- Cons
  - “value” of Properties and “object” of Relationships can be either Object or Array

  → needs to be checked when parsing

Change: Query Language Syntax Changes to Attribute Path

Attribute Path representation in Query Language

- For better readability, we changed the attribute path representation in the Query Language.
- The attribute path is used in the query language when comparing properties/relationships of properties/relationships and elements of values with specific simple values, respectively.
- Thus it is necessary to distinguish whether a name refers to e.g. a property of property or an element of a value.
- In the new version: only when first referring to an element of a value, square brackets are used – inside the value the dot notation is used in the same way as when referring to (e.g.) properties of properties. → see examples on next slide

Attribute Path Example

```
{
 "id": "urn:ngsi-ld:Piece:A4567",
 "type": "Piece",
 "component": {
          "type": "Property",
          "element": {
                    "type": "Property",
                     "subelement": {
                              "type": "Property",
                              "value": "subelement_value"
                    }
          },
          "value": {
                    "element": {
                               "subelement": "subelement_value"
                    }
          }
  }
}
```

- To compare the value of the subelement property of the element property of the component property, the path would have to be:

```
component.element.subelement="subelement_value"
```

- To compare the subelement of element of the value of the component property, the path would have to be:

```
component[element.subelement]="subelement_value"
```

Comparison of current and previous definition

New definition:

```
DottedPath = AttrName *(%x2E AttrName)                  ; AttrName *(.AttrName)
Attribute = DottedPath *1(%x5B DottedPath %x5D)         ; DottedPath *1([DottedPath])
```

Example: `property.property[value_level1.value_level2.value_level3]`

Previous definition:

```
attrPathName = attrName *(%x2E attrName)                    ; attrName *(. attrName)
compoundAttrName = attrPathName *(%x5B attrName %x5D)       ; . attrName *([ attrName ])
```

Example: `property.property[value_level1][value_level2][value_level3]`

Change: Batch Operation Response Codes

HTTP Binding of Batch Operations: CIM009v1.3.1 Clauses 6.14, 6.15, 6.16, 6.17

Batch Operation Response Codes

We adapted the Batch Operations Response Codes to better align with the non-Batch versions and to better reflect the cases where the operation was successful only for some of the individual requests in the batch (207).

Code	Create	Upsert	Update	Delete
201 Created	If all entities have been successfully created.	If all entities not existing prior to request have been successfully created and others successfully updated.		
204 No content		If all entities previously existed and are successfully updated.	If all entities have been successfully updated.	If all entities already existed and have been successfully deleted.
207 Multi Status	If only some entities have been successfully created.	If only some or none of the entities have been successfully created or updated.	If only some or none of the entities have been successfully updated.	If some or all of the entities have not been successfully deleted, or did not exist.
400 Bad Request	If the request or its content is incorrect.	If the request or its content is incorrect.	If the request or its content is incorrect.	If the request or its content is incorrect.

102r1 in 6.17 has not been completely implemented

Feature: Query for available Entity Types and Attributes

Feature “Query for Available Entity Types and Attributes”

- It is important to be able to find out what kind of information is currently available in an NGSI-LD system
- In NGSI-LDv1.2.1 only the request for entities based on specific entities, entity types and attributes was available in NGSI-LD
- **Now there are requests to retrieve currently available Entity Types or currently available Attributes**

Retrieve Available Entity Types: CIM009v1.3.1 Clauses 5.7.5, 5.2.24, 6.25

Retrieve Available Entity Types

- Resource: /ngsi-ld/v1/types
  - HTTP GET
    - id: urn:ngsi-ld:EntityTypeList:<unique>
    - JSON-LD object of Type EntityTypeList
    - typeList element with Array of Entity Type Names (value of type @vocab, to enable expansion / compaction)

```
{
        "id": "urn:ngsi-ld:EntityTypeList:1236645",
        "type": "EntityTypeList",
        “typeList": [
                  "https://uri.etsi.org/ngsi-ld/primer/Shelf",
                  "https://uri.etsi.org/ngsi-ld/primer/Store"
         ]
}
```

with @context in link header:

```
{
        "id": "urn:ngsi-ld:EntityTypeList:34534657",
        "type": "EntityTypeList",
        “typeList": [
                  “Shelf",
                  "Store"
         ]
}
```

Retrieve Details of Available Entity Types: CIM009v1.3.1 Clauses 5.7.6, 5.2.25, 6.25

Retrieve Details of Available Entity Types

- Resource: /ngsi-ld/v1/types?options=details
  - HTTP GET
    - Array of Entity Type Information
      - id: URI of entity type being described
      - type: "EntityType“
      - typeName: name of entity type @vocab, to enable expansion / compaction
      - attributeNames: array of attribute names

with @context in link header:

```
[
  {
        "id": "https://uri.etsi.org/ngsi-ld/primer/Store"
        "type": "EntityType",
        "typeName": "Store“,
        "attributeNames": [“storeName", "address", "location",
            “contains"]
   },
   {
        "id": "https://uri.etsi.org/ngsi-ld/primer/Shelf",
        "type": "EntityType",
        "typeName": “Shelf",
        "attributeNames": [“maxCapacity", "location",
            “isContainedIn"]
   }
]
```

Retrieve Available Entity Type Information: CIM009v1.3.1 Clauses 5.7.7, 5.2.26, 6.26

Retrieve Available Entity Type Information

- Resource: /ngsi-ld/v1/types/{entity type name}
  - HTTP GET
    - Detailed Entity Type Information
      - id: name of entity type being described
      - type: "EntityType“
      - typeName: name of entity type @vocab, to enable expansion / compaction
      - entityCount: number of entity instances that have this type
      - attributeDetails: array of attribute details
        - id: URI of attribute being described
        - type: "EntityAttribute“
        - attributeName: name of attribute @vocab, to enable expansion / compaction
        - attributeTypes: array of attribute types with which attribute instances exist (often, but not always one)

with @context in link header:
/ngsi-ld/v1/types/Store

```
{
        "id": "https://uri.etsi.org/ngsi-ld/primer/Store",
        "type": "EntityTypeInformation",
        "typeName": "Store“,
        “entityCount“: 4,
        "attributeDetails": [ {
                "id": "https://uri.etsi.org/ngsi-ld/location",
                "type": "Attribute",
                "attributeName": "location",
                "attributeTypes": ["GeoProperty"],
             },
        …]
 }
```

Retrieve Available Attributes: CIM009v1.3.1 Clauses 5.7.8, 5.2.27, 6.27

Retrieve Available Attributes

- Resource: /ngsi-ld/v1/attributes
  - HTTP GET
    - id: urn:ngsi-ld:AttributeList:<random>
    - JSON-LD object of Type AttributeList
    - attributeList element with Array of Attribute Names (value of type @vocab, to enable expansion / compaction)

```
{
        "id": "urn:ngsi-ld:AttributeList:1236645",
        "type": “AttributeList",
        “attributeList": [
                "https://uri.etsi.org/ngsi-ld/primer/contains",
                "https://uri.etsi.org/ngsi-ld/primer/isContainedIn",
                "https://uri.etsi.org/ngsi-ld/location"]
}
```

with @context in link header:

```
{
        "id": "urn:ngsi-ld:AttributeList:7896645",
        "type": “AttributeList",
        “attributeList": [
                "contains",
                "isContainedIn",
                "location"]
}
```

Retrieve Details of Available Attributes: CIM009v1.3.1 Clauses 5.7.9, 5.2.28, 6.27

Retrieve Details of Available Attributes

- Resource: /ngsi-ld/v1/attributes?options=details
  - HTTP GET
    - Array of Attribute Information
      - id: URI of attribute being described
      - type: "Attribute"
      - attributeName: name of attribute @vocab, to enable expansion / compaction
      - typeNames: array of entity types that have an attribute of this name, @vocab, to enable expansion / compaction

with @context in link header:

```
[
    {
        "id": " https://uri.etsi.org/ngsi-ld/primer/contains",
        "type": "Attribute",
        "attributeName":  "contains",
        "typeNames": ["Store"]
     },
     {
        "id": "https://uri.etsi.org/ngsi-ld/location",
        "type": "Attribute",
        "attributeName":  "location",
        "typeNames": [
               "Store",
               "Shelf"]
     },
     …
]
```

Retrieve Available Attribute Information: CIM009v1.3.1 Clauses 5.7.10, 5.2.28, 6.28

Retrieve Available Attribute Information

- Resource: /ngsi-ld/v1/attributes/{attribute name}
  - HTTP GET
    - Detailed Attribute Information
      - id: URI of attribute being described
      - type: "Attribute"
      - attributeName: name of attribute @vocab, to enable expansion / compaction
      - attributeTypes: array of attribute types with which attribute instances exist
      - typeNames: array of entity types that have an attribute of this name,

        @vocab, to enable expansion / compaction
      - attributeCount: number of attribute instances that have this name

with @context in link header:
/ngsi-ld/v1/attributes/location

```
{
        "id": "https://uri.etsi.org/ngsi-ld/location",
        "type": "Attribute",
        "attributeName":  "location",
        "attributeTypes": ["GeoProperty"],
        "typeNames": [
                "Store",
                "Shelf"]
        "attributeCount": 5
}
```

Feature: Full GeoJSON documents as query responses and notifications

GeoJSON – Motivation

- I want to ease the path of adoption for NGSI-LD for Context Information for developers
- As a developer, I want to display Context Information on a Map.
- Many GIS Systems already support a common standard for loading data layers
- Therefore, I want to be able to make queries to an NGSI-LD system and retrieve responses in a **GeoJSON / GeoJSON-LD** friendly format.

- The GeoJSON specification is **RFC 7946** - see: https://tools.ietf.org/html/rfc7946
  The GeoJSON-LD **@context** can be found here: http://geojson.org/geojson-ld/
- Typically used for Geographic Information
- Each NGSI-LD entity is equivalent to a GeoJSON **Feature**
- GeoJSON is already used for **GeoProperties** within NGSI-LD

```
{
   "type": "Feature",
   "id": "f1",
   "geometry": {...},
   "properties": {...}
}
```

Solution - Output NGSI-LD Entities as GeoJSON Features

NGSI-LD entity:

```
{
   "@context": "...etc",
   "id": "urn:ngsi-ld:Building:store001",
   "type": "Building",
   "address": {
      "streetAddress": "Bornholmer Straße 65",
      "addressRegion": "Berlin",
      "addressLocality": "Prenzlauer Berg",
      "postalCode": "10439"
   },
   "name": "Bösebrücke Einkauf",
   "category": "commercial",
   "location": {
      "type": "Point",
      "coordinates": [
         13.3986,
         52.5547
      ]
   }
}
```

GeoJSON Feature

```
{
     "type": "Feature",
     "id": "urn:ngsi-ld:Building:store001",
     "geometry": {
       "type": "Point", "coordinates": [13.3903, 52.5075]
     },
     "properties": {
       "type" : "Building",
       "name": "Checkpoint Markt",
       "category": "commercial",
       "address": {
          "streetAddress": "Friedrichstraße 44",
          "addressRegion": "Berlin",
          "addressLocality": "Kreuzberg",
          "postalCode": "10969"
       }
     },
     “@context”:  “...etc”
   }
```

- Basically all we have done here is:
- Renamed and the expanded "location" => "geometry"
- Relegated the NGSI-LD Entity "type" and placed it into "properties"

How do GeoJSON and NGSI-LD Align?

- GeoJSON "type" is always "Feature" for a single NGSI-LD entity, or "FeatureCollection" for arrays of NGSI-LD entities.
  - GeoJSON "type" is mandated to be one of 7 basic types - NGSI-LD only needs two.
- GeoJSON "id" is the direct equivalent NGSI-LD Entity "id"
- GeoJSON "geometry" is direct equivalent of an NGSI-LD Entity's GeoProperty, the default is "location"
  - "geometry" is never key-value pairs - always expanded including "coordinates".
  - NGSI-LD GeoProperty accepts a subset of valid GeoJSON geometry types anyway so will always be a valid "geometry".
  - If an entity is requested without a "location" then passing null is valid
- GeoJSON "properties" are free-form JSON attributes
  - holds the Entity type" and all the remaining NGSI properties and relationships
  - The name "properties" clashed with "properties" defined for Subscriptions (same with Subscription "title" and "description") in NSGI-LD v1.2.1 and thus the NGSI-LD term was renamed to "propertyNames" in NSGI-LD v1.3.1 (see slide 6).

GeoJSON documents: CIM009v1.3.1 especially Clauses 4.5.16, 4.5.17, 5.7.1, 5.7.2, 6.3.15

Support of GeoJSON documents in NGSI-LD v1.3.1

- GeoJSON documents can only be requested, i.e. be used only for retrieval and query responses as well as notifications resulting from subscriptions
- To retrieve a GeoJSON document, the accept header has to be set to Accept: application/geo+json
- To get the @context as part of the body: Omit Prefer Header or set it to "body=ld+json"
- To get the @context as a Link Header: Set Prefer Header to "body=json“.
- In HTTP GET operations set the geometryProperty to the name of the GeoProperty to be used as GeoJSON Geometry – by default location is used. Optionally, a datasetId can be specified as an additional parameter.

Examples for Requesting GeoJSON Documents

GET /ngsi-ld/v1/entities/urn:ngsi-ld:xxx
Accept: application/geo+json

```
{
    "type": "Feature",
    "id": "urn:ngsi-ld:Building:store002",
    "geometry": {
      "type": "Point", "coordinates": [13.3903, 52.5075]
    },
    "properties": {
      "type" : "Building",
      "address": {
          "type": "Property",
          "value": {
              "streetAddress": "Bornholmer Straße 65",
              "addressRegion": "Berlin",
              "addressLocality": "Prenzlauer Berg",
              "postalCode": "10439"
          },
          "verified": {
              "type": "Property",
              "value": true
          }
      },
      "name": {
          "type": "Property",
          "value": "Bösebrücke Einkauf"
      },
      "category": {
          "type": "Property",
          "value": "commercial"
      },

    },
   "@context": "...etc"
}
```

GET /ngsi-ld/v1/entities/urn:ngsi-ld:xxx?options=keyValues
Accept: application/geo+json

```
{
    "type": "Feature",
    "id": "urn:ngsi-ld:Building:store002",
    "geometry": {
      "type": "Point", "coordinates": [13.3903, 52.5075]
    },
    "properties": {
      "type" : "Building",
      "name": "Checkpoint Markt",
      "category": "commercial",
      "address": {
          "streetAddress": "Friedrichstraße 44",
          "addressRegion": "Berlin",
          "addressLocality": "Kreuzberg",
          "postalCode": "10969"
      }
    }
  },
  "@context": ""...etc"
}
```

Feature: MQTT Notification Binding

Motivation for Having MQTT Notifications

- Using REST for notifications, thus
  - opening a new connection to the notification receiver for each notification
  - sending all HTTP headers over and over again
- … is **slow**
- A permanent connection makes a huge impact on performance
- MQTT publishers and subscribers open permanent connections to an MQTT Server/Broker

Design 1

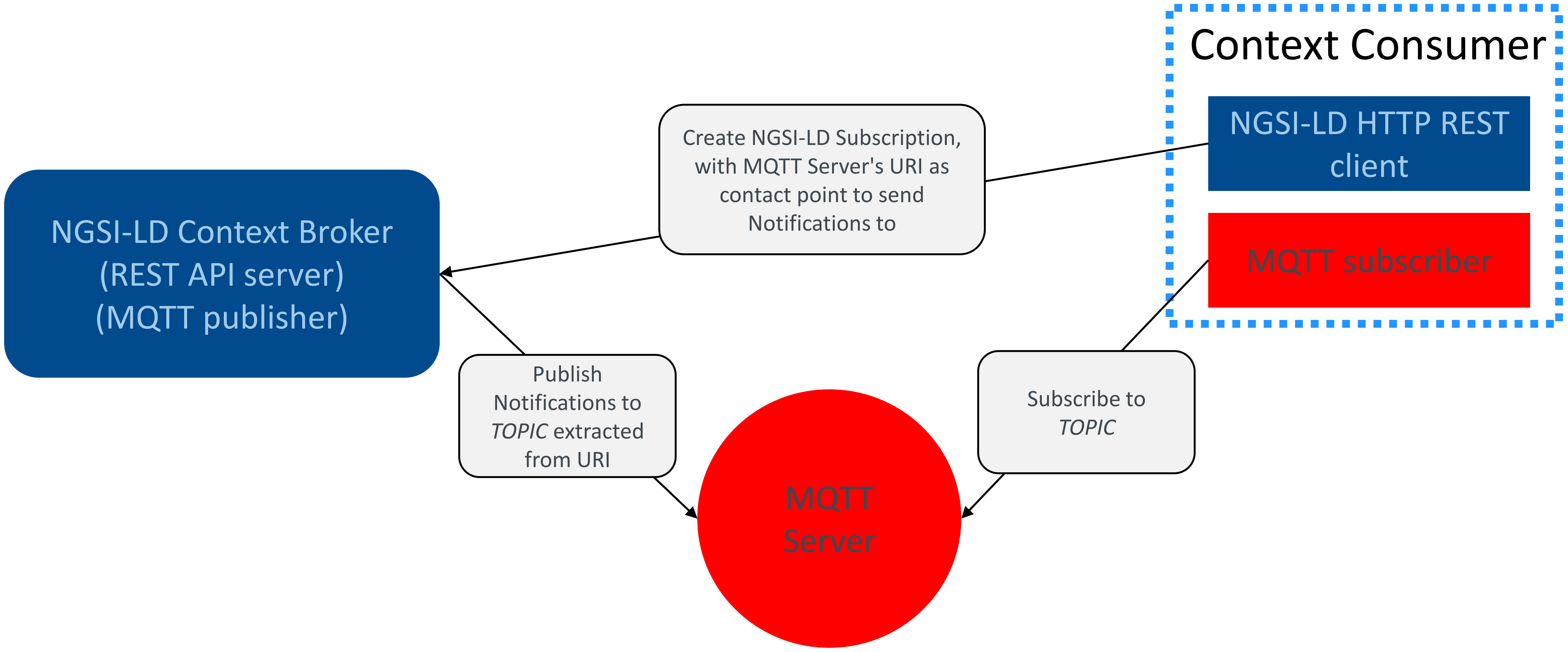
- MQTT in NGSI-LD (upcoming v1.3.1) is **only used for sending notifications**, not for requesting them (i.e. **not** for creating an NGSI-LD Subscription)
  - Full support of MQTT in future release is planned
- NGSI-LD Subscriptions support the following **MQTT URI structure**, to reach back to the client
  - mqtt[s]://[<username>][:<password>]@<host>[:<port>]/<topic>[/<subtopic>]
- MQTT, unlike HTTP, does not have a headers vs. body structure
  - NGSI-LD Context Broker generates **notifications as JSON Objects**
  - "metadata" is an object with {key, value} pairs, it contains the header info
  - "body" contains the payload data of the notification itself

```
{
  "metadata":{
    "<key>":"<value>",
    …
  },
  "body":{
    <body>
  }
}
```

Design 2

- The client wanting to receive Notifications, **creates the corresponding Subscription** in the Context Broker, via the usual REST HTTP NGSI-LD API
- The Subscription contains **the Notification parameters** that Context Broker uses to deliver matching Notifications back to the client
- The MQTT URI (mqtt or mqtts) provided in the Notification parameters is the **contact point** of an MQTT Server
- Context Broker acts as publishing client of the MQTT Server
  - It publishes the JSON Object to the **topic extracted from** the MQTT URI provided in the Notification parameters
- Subscribers of the same topic (clients of the MQTT Server) receive the **stream of notifications**, which contain payload data and headers

Architecture



Context Consumer
NGSI-LD HTTP REST client
MQTT subscriber
Create NGSI-LD Subscription, with MQTT Server's URI as contact point to send Notifications to
NGSI-LD Context Broker (REST API server) (MQTT publisher)
Publish Notifications to *TOPIC* extracted from URI
MQTT Server
Subscribe to *TOPIC*

Implementation: Endpoint Specification within Notification Parameters

- Endpoint-related parameters are required when creating NGSI-LD Subscriptions
- They define an endpoint to send back notifications to

Parameter	Data type	Restrictions	Cardinality	Description
uri	URI	Dereferenceable URI	1	URI which conveys the endpoint which will receive the notification
accept	string	MIME type. It shall be one of: "application/json" "application/ld+json" "application/geo+json"	0..1	Intended to convey the MIME type of the notification payload body (JSON or JSON-LD)
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

Implementation:
MQTT Parameters go into *Endpoint.notifierInfo*

- MQTT supports two versions based on OASIS standards (v3.1.1 and v5.0)
- MQTT has three levels of QoS (0, 1, 2)
- *Endpoint.notifierInfo*, in case of MQTT binding, holds the configuration that the broker needs to know to correctly communicate with receiver (MQTT-Version, MQTT-QoS)
- *Endpoint.notifierInfo* is an array of key-value pairs
- MQTT parameters are specified as keys and values in *Endpoint.notifierInfo*, as follows

Key	Possible Values
MQTT-Version	mqtt3.1.1, mqtt5.0
MQTT-QoS	0, 1, 2

Implementation: MQTT "metadata" Object in Published Notification JSON

- NGSI-LD Context Broker includes the following additional information into the "metadata" of the published MQTT JSON message
  - MIME type
  - link to the @context (if needed)
  - additional user-specified information
- When REST Notifications are used instead of MQTT Notifications, this information is provided as HTTP headers

Implementation: MQTT "metadata" Object in Published Notification JSON

- The MIME type is specified as "Content-Type" key in the "metadata" object of the MQTT JSON message
  - The MIME type associated with the notification shall be "application/json" by default
  - This can be changed to application/ld+json by means of *Endpoint.accept* member
- If the target MIME type is "application/json" then the reference to the JSON-LD @context is provided as "Link" key

Key	Possible Values	Default	Description
Content-Type	application/json, application/ld+json	application/json	MIME type of the notification included in the "body" element of the MQTT message
Link	Same format as specified in JSON-LD specification, clause 6.8, for the HTTP Link header		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"

- If the optional {key, value} pairs array *Endpoint.receiverInfo* is present in the notification parameters of the original subscription, **then a new entry for each additional key shall be added to the "metadata"**

Feature: COUNT header option

Count: CIM009v1.3.1 Clauses 4.13, 6.3.13

Counting query results

- Requests to the NGSI-LD API can return a large number of results
- In NGSI-LD v1.3.1 it is possible to request the overall *count* of results, even if paging functionality is used and only few results are returned. By setting the paging *limit* to 0 only the count is returned. A limit of 0 is only allowed in combination with requesting count.
- Query operations based on HTTP GET support the query parameter *count* (boolean). If set to *true*, the response includes the special HTTP header (NGSILD-Results-Count) with the count of the overall number of available results as value.

Feature: Multi-Tenant Support

Multitenancy

General Definition:
The term "**software multitenancy**" refers to a software architecture in which a single instance of software runs on a server and serves multiple tenants. Systems designed in such manner are often called **shared** (in contrast to **dedicated** or **isolated**). A **tenant is a group of users** who **share a common access** with specific privileges to the software instance. With a multitenant architecture, a software application is designed to **provide every tenant a dedicated share of the instance** - including its **data, configuration, user management, tenant individual functionality and non-functional properties**. Multitenancy contrasts with multi-instance architectures, where separate software instances operate on behalf of different tenants.
https://en.wikipedia.org/wiki/Multitenancy

- In the context of NGSI-LD: Multiple tenants (e.g. users, groups of users or organizations) should be able to use the same NGSI-LD (Broker) deployment
  - Tenants need to be isolated from each other:
    - Information (entities, properties, relationships)
    - subscriptions
    - registrations

    of one tenant need to be “invisible” to other tenants
  - From API perspective:
    - It is required to identify tenants
    - Use of tenants needs to be clear across distributed and federated deployments

Side note: Multi-tenancy is closely linked to access control, i.e. which tenant may access what information

Multi-tenancy: CIM009v1.3.1 Clauses 4.14, 5.5.10, 6.3.14

Solution Summary

- NGSI-LD implementations can **optionally support multitenancy.** In all requests **a tenant can optionally be specified**. In the HTTP binding this is done through the **HTTP header "NGSILD-Tenant"**
- The support for *tenants* can be **implicit**, i.e. a tenant is created when first used in the context of a create (entity, subscription, registration) operation. There are currently no API operations for explicitly creating or deleting tenants.
- Principle: **if no tenant is specified**, there is always a "**default tenant**" (which does not have to be created and does not necessarily have to have any information to provide).
- How tenants are implemented, i.e. how isolation is achieved, is up to the implementation.
- **Registrations target a tenant (if not, the "default tenant" is assumed)** – if the same context source / broker is to be registered for multiple tenants, multiple registrations are needed.
- Registrations can provide tenant information. If a context source / broker is registered with certain tenant information, this tenant information is used for forwarding the request, not the provided tenant information in the original request (if no tenant information is provided, the "default tenant" will be assumed), thus the registry enables mapping of tenant information.

Tenants in Distributed/Federated Cases: Tenant Mapping in the Registry

- As for other operations, **tenant can be specified in registry operations**
- Due to isolation between tenants, there are de facto **separate registries per tenant** (realization is implementation-dependent)
- The available **tenants in registered Context Sources or Brokers may be different from those in the Broker** to which the registry belongs → mapping is required
- Thus: optional tenant information can be added to registrations (see next slide)
- Brokers then **use this tenant information from the registrations when interacting with the registered Context Sources / Brokers** (not the tenant information provided in the original request)

Multi-tenancy: CIM009v1.3.1 Clauses 4.14, 5.5.10, 6.3.14 & **5.2.9**

Tenant Information in CsourceRegistrations

Name	Data type	Restriction	Cardinality	Description
id	URI	At creation time, If it is not provided, it will be assigned during registration process and returned to client. It cannot be later modified in update operations	0..1	Unique registration identifier. (JSON-LD @id). There may be multiple registrations per Context Source, i.e. the id is unique per registration
type	string	"ContextSource Registration"	1	JSON-LD @type Use reserved type for identifying Context Source Registration
name	string	Non-empty string	0..1	A name given to this Context Source Registration
description	string	Non-empty string	0..1	A description of this Context Source Registration
Information	RegistrationInfo[]	See data type definition in clause 5.2.10. 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	**URI**		**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.**
[…]	[…]	[…]	[…]	[…]

Feature: POST Queries

POST Queries: CIM009v1.3.1 Clause 6.23

Motivation

The reason to provide a way to query NGSI-LD entities via POST (in addition to the proper RESTful way that uses GET), is that, using GET:

1. 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. (See discussion on this in StackOverflow).
2. There is a need to URL-encode the resulting URL. By using POST, there’s no need to url-encode

The difference lies in that instead of passing the inputs as URI parameters (as for the GET service), for the POST Query service, the user passes all the query items in the payload

Design

We created a novel **/entityOperations/query** endpoint in the API

The new POST query operation has just one single query as input

- It is not allowed to send multiple queries in a single request
- It is not to be considered a "BATCH Operation"

The problem with having multiple queries in the request payload body (as an array) is having a different @context in each item of the array (for Content-Type: application/ld+json)

- would greatly complicate the response
- we simply decided not to allow it

Data Type of the Request Payload Body (with references to relevant NGSI-LD API clauses)

Name	Data type	Restrictions	Cardinality	Description
type	string	It shall be equal to "Query"	1	JSON-LD @type
entities	EntityInfo[]	See data type definition on clause 5.2.8. 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 on clause 5.2.13	0..1	Geo-Query 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 on clause 5.2.21	0..1	Temporal Query to be present only for “Query Temporal Evolution of Entities” operation (clause 5.7.4)

Request URL and Example Payload Data

```
POST /ngsi-ld/v1/entityOperations/query
{
  “type”: “Query”,
  “entities”: [
    {
      “id”: “urn:...”,
      “type”: “”,
      “idPattern”: “”
    },
    {}, ...
  ],
  “attrs”: [ “P1”, “P2”, “R1”, “R2” ],
  “q”: “P1.x<12”,
  “geoQ”: {
    “geometry”: “Point”,
    “coordinates”: [ 1.0, 2.0 ],
    “georel”: “near;maxDistance==5000”,
    “geoproperty”: “loc”
  },
  “csf”: “xxx”,
  “temporalQ”: {
    “timerel”: “”,
    “timeAt”: “”,
    “endTimeAt”: “”,
    “timeProperty”: “”
  }
}
```

Feature: Specification of additional headers for notifications

Notification Parameters

- Additional parameters for notifications can be specified as key-value pairs in subscriptions.
- Parameters required on the receiver side are specified in receiverInfo (e.g. authentication parameters), parameters required on the sender side in notifierInfo (e.g. protocol or quality information)
- In HTTP, receiverInfo is sent as HTTP headers, in MQTT as metadata in the body of the message.

Parameter	Data type	Restrictions	Cardinality	Description
uri	URI	Dereferenceable URI	1	URI which conveys the endpoint which will receive the notification
accept	string	MIME type. It shall be one of: "application/json" "application/ld+json" "application/geo+json"	0..1	Intended to convey the MIME type of the notification payload body (JSON or JSON-LD)
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

Feature: Support for queries where attribute is implicitly specified in query filter or geoquery

Queries: CIM009v1.3.1 Clause 5.7.2

Support for queries where attribute is implicitly specified

- In NGSI-LD v1.2.1 only queries are allowed where at least either an entity type or an attribute is explicitly specified
- This has been extended in NGSI-LD v1.3.1 to also allow queries where only a (filter) query or a geoquery is specified. These restrict/filter according to attributes which can be used instead of explicitly specified attribute(s) to find relevant entities.

“It is not possible to retrieve a set of entities by only specifying desired 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 geo-query.”

Feature: Support for international characters in URIs/IRIs and terms

International Characters: CIM009v1.3.1 Clauses 4.6.2, 5.2.1 (IRIs)

Support for international characters in URIs/IRIs and terms

- In NGSI-LD v1.3.1 all legal IRIs are supported, in previous versions only ASCII-based URIs
- In the terms all Unicode characters of the number and letter category (plus the underscore) are supported, in previous versions it was only ASCII letters and numbers. This allows the use of different scripts for terms, e.g. Japanese, Korean, Chinese, Hindi, …

```
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}`.