ETSI GS CIM 047 V1.1.2 (2024-12)

Context Information Management (CIM); OpenAPI Specification for NGSI-LD API

Disclaimer

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

Reference
RGS/CIM-0047v112

Keywords
API, NGSI-LD

ETSI

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

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

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

Contents

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

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

Modal verbs terminology

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

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

Executive summary

The present document formally describes the OpenAPI Specification for the NGSI-LD API specified by ETSI GS CIM 009 [1], [2] and [3]. The OpenAPI Specification allows users to make use of the NGSI-LD API in a language-agnostic form. With a declarative resource specification followed by the OpenAPI Specification, NGSI-LD API clients can understand and consume services without knowledge of the server-side implementation.

The present document outlines the design and evaluation strategies for the OpenAPI Specification implementation, as well as its general structure and content. Practical examples are also included throughout the present document to help readers understand the usability of the OpenAPI.

Introduction

The present document defines the OpenAPI Specification for the standard NGSI-LD API for Context Information Management. OpenAPI is a popular standard for building REST APIs independently of the implementation language. Having an OAS for the NGSI-LD API helps users by facilitating API documentation and allowing them to implement and use the NGSI-LD protocol in their own application.

To this end, the present document aims to provide information to help better understand the implementation and application details of the OpenAPI Specification for the NGSI-LD API. The implementation of the OpenAPI Specification is based on the clauses defined for the NGSI-LD API considered in ETSI GS CIM 009 [1], [2] and [3].

1 Scope

The purpose of the present document is the definition of the OpenAPI Specification for the standard NGSI-LD API for Context Information Management. Documentation will be provided, along with examples that help developers and users understand how the OpenAPI works and how to use it in a programming language-agnostic form.

2 References

2.1 Normative references

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

Referenced documents which are not found to be publicly available in the expected location might be found at ETSI docbox.

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

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

[1] ETSI GS CIM 009 (V1.6.1): "cross-cutting Context Information Management (CIM); NGSI-LD API".

[2] ETSI GS CIM 009 (V1.7.1): "Context Information Management (CIM); NGSI-LD API".

[3] ETSI GS CIM 009 (V1.8.1): "Context Information Management (CIM); NGSI-LD API".

[4] OpenAPI Specification (v3.0.3).

[5] Swagger Documentation - OpenAPI Specification (v3.0.3).

[6] OpenAPI Specification (v3.1.0).

[7] NGSI-LD OAS release for NGSI-LD API version 1.6.1.

[8] NGSI-LD OAS release for NGSI-LD API version 1.7.1.

[9] NGSI-LD OAS release for NGSI-LD API version 1.8.1.

[10] IETF RFC 7807: "Problem Details for HTTP APIs".

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

2.2 Informative references

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

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

The following referenced documents are not necessary for the application of the present document, but they assist the user with regard to a particular subject area:

[i.1] OpenAPI Initiative.

[i.2] OpenAPI Tooling.

[i.3] OpenAPI Documentation.

[i.4] OpenAPI Tooling Categories.

[i.5] OpenAPI descriptions for the NGSI-LD API defined by ETSI ISG CIM.

[i.6] OpenAPI Swagger Editor Extension in VS Code.

[i.7] Swagger Editor: API editor for designing APIs with the OpenAPI and AsyncAPI specifications.

[i.8] Swagger UI: Visualize OpenAPI Specification definitions in an interactive UI.

[i.9] ReDoc: Generate beautiful API documentation from OpenAPI.

[i.10] Scalar API Reference: Beautiful API references from OpenAPI/Swagger files.

[i.11] OpenDocumenter: Automatic documentation generator for OpenAPI v3 schemas.

[i.12] API Security Audit.

[i.13] Swagger 2.0 and OpenAPI 3.0 parser/validator.

[i.14] Express OpenAPI Validator: Auto-validates api requests, responses, and securities using ExpressJS and an OpenAPI 3.x specification.

[i.15] Swagger UI - NGSI-LD OAS release for NGSI-LD API version 1.7.1.

[i.16] Redocly - NGSI-LD OAS release for NGSI-LD API version 1.7.1.

[i.17] OpenAPI Generator: Generate clients, servers, and documentation from OpenAPI 2.0/3.x documents.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

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

NGSI-LD Context Broker: architectural component that implements all the NGSI-LD interfaces

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

NGSI-LD Entity Type: categorization of an NGSI-LD Entity as belonging to a class of similar entities, or sharing a set of characteristic properties

NGSI-LD GeoProperty: subclass of NGSI-LD Property which is a description instance which associates a main characteristic, i.e. an **NGSI-LD Value**, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property, that uses the special *hasValue* property to define its target value and holds a geographic location in GeoJSON format

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

NGSI-LD ListProperty: description instance which associates an ordered array of main characteristics, i.e. **NGSI-LD Values**, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property and that uses the special *hasValueList* property to define its target value

NGSI-LD ListRelationship: description of an ordered array of directed links between a subject which is either an NGSI-LD Entity, an NGSI-LD Property or another NGSI-LD Relationship on one hand, and a series of objects, which are NGSI-LD Entities, on the other hand, and which uses the special *hasObjectList* property to define its target objects

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

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

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

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

NGSI-LD Value: JSON value (i.e. a string, a number, *true* or *false*, an object, an array), or JSON-LD typed value (i.e. a string as the lexical form of the value together with a type, defined by an XSD base type or more generally an IRI), or JSON-LD structured value (i.e. a set, a list, a language-tagged string)

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

3.2 Symbols

Void.

3.3 Abbreviations

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

API	Application Programming Interface
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
JSON-LD	JSON Linked Data
NGSI	Next Generation Service Interfaces
NGSILD	Next Generation Service Interfaces Linked Data (same as NGSI-LD)
NGSI-LD OAS	NGSI-LD OpenAPI Specification
MIME	Multi-purpose Internet Mail Extensions
OAS	OpenAPI Specification
RFC	Request For Comments
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XSD	XML Schema Definition

4 OpenAPI Specification for NGSI-LD API

4.1 Introduction

OpenAPI is the de-facto standard for building REST APIs in a programming language-agnostic way. OpenAPI specifications can improve the documentation of the APIs, and they can also be used to generate stub code for clients and servers in multiple programming languages. An OpenAPI Specification (OAS) for NGSI-LD API is useful for developers to implement the NGSI-LD API in their applications.

4.2 Design, evaluation, and structure of OpenAPI Specification

4.2.0 Foreword

This clause describes the technical design and evaluation principles behind the NGSI-LD OpenAPI Specification, hereinafter referred to as NGSI-LD OAS. In addition, this clause provides an overview of the main structure for the definition of the NGSI-LD OAS.

4.2.1 Design strategy

Figure 4.2.1-1 depicts the design workflow followed to complete the NGSI-LD OAS implementation. The goal is to design a new NGSI-LD OAS for each new version of the NGSI-LD API specification (i.e. each new release of the ETSI GS CIM 009 [1], [2] and [3]).

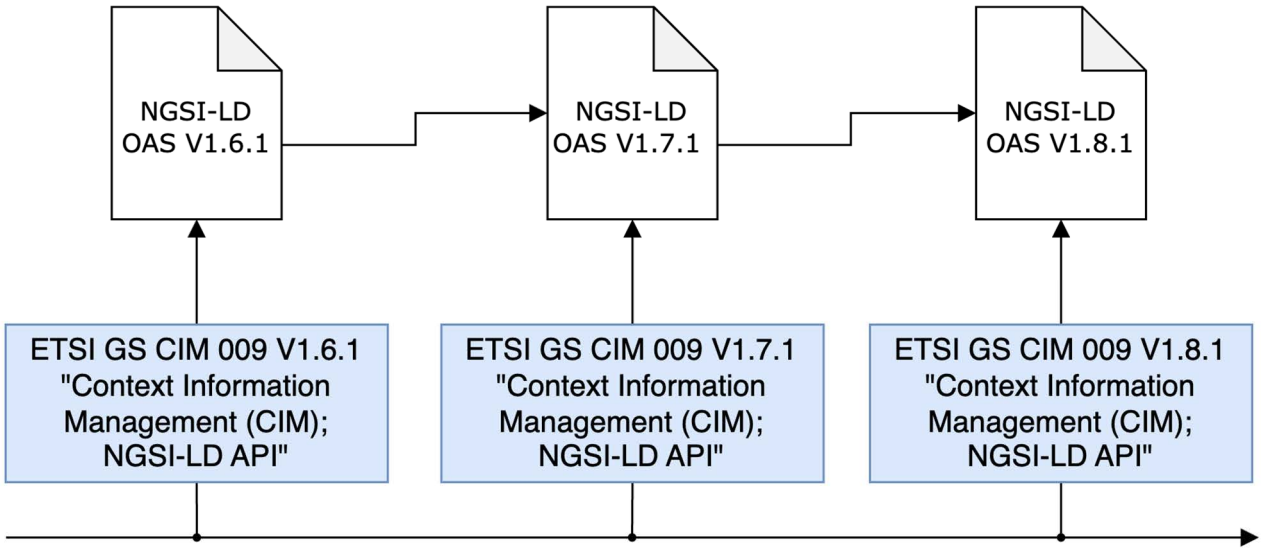

Figure 4.2.1-1: Design workflow for the NGSI-LD OAS

The development of the NGSI-LD OAS is incremental, using a stable version of it as the basis for the next version. The starting point is a first release of the NGSI-LD OAS implemented for version 1.6.1 of the NGSI-LD API [1], which is used as the baseline for the following releases. Thus, when defining the NGSI-LD OAS for a new version of the NGSI-LD API, the strategy is to complement the design of the NGSI-LD OAS definition for the previous version of the NGSI-LD API with the advances included within the specification document of the new version for the NGSI-LD API.

In the process of defining the OAS, ensuring a clear structure is essential. The organization and description of the NGSI-LD OAS structure is detailed in clause 4.2.3. Open-source tools used to facilitate the design and development of the NGSI-LD OAS are presented in clause A.2 of Annex A.

4.2.2 Evaluation strategy

Once the NGSI-LD OAS is defined, it is necessary to evaluate it to ensure that it works as expected. This evaluation consists of the following stages:

- validating the structure of the OAS, mainly verifying the definition of schemas and operations;
- verifying the capabilities of the OAS to interact with servers that implement the NGSI-LD API;
- generating stub code usable in external applications.

For the validation of the schemas and operation specified within the NGSI-LD OAS, different type of OpenAPI tools have been used as represented in Figure 4.2.2-1 [i.1] and [i.2]. Some of the open-source tools to facilitate the evaluation of the NGSI-LD OAS are presented in clause A.3 of Annex A. Figure 4.2.2-1 also shows the type of tools used to facilitate the edition and auditability of OAS implementation that are useful for the design strategy. For the creation of the NGSI-LD OAS, the developer carries out the design and evaluation processes continuously, using the aforementioned utilities.

To verify if the API operations defined within the NGSI-LD OAS work, there are tools to navigate the OAS operations (i.e. the OpenAPI Visualization Tool in Figure 4.2.2-1) and test them directly against servers that implement the NGSI-LD API such as NGSI-LD compliant Context Brokers (i.e. the OpenAPI Interaction Tool in Figure 4.2.2-1). Thus, these visualization and interaction tools help to verify the documentation and execution of the different NGSI-LD API operations defined in the NGSI-LD OAS itself, checking the meaning and functionality of each of the parameters and options specified by each operation and determining whether they are well established within the NGSI-LD OAS or whether the NGSI-LD Context Broker implementations support them accordingly. Annex B provides examples of such open-source tools that can be used to visualize and interact with the API resources defined within an OAS.

As part of the evaluation strategy, testing NGSI-LD OAS stub code generated for clients against NGSI-LD Context Brokers compliant with the respective version of the NGSI-LD API is another approach to evaluate the capabilities of the NGSI-LD OAS. Annex D shows examples for generating stub code from NGSI-LD OAS for NGSI-LD clients in different programming languages, as well as different examples of use. Moreover, Figure 4.2.2-1 depicts a complete evaluation workflow for testing the NGSI-LD OAS using client-side libraries automatically generated from the OpenAPI specification itself. The workflow considers that developers might define OpenAPI schemas compliant with the NGSI-LD OAS for their specific purpose applications in order to define particular NGSI-LD information models in a programming code-based form that could be used directly from the generated NGSI-LD clients. Annex C provides more information and guidelines about how to define custom schemas compliant with the NGSI-LD OAS.

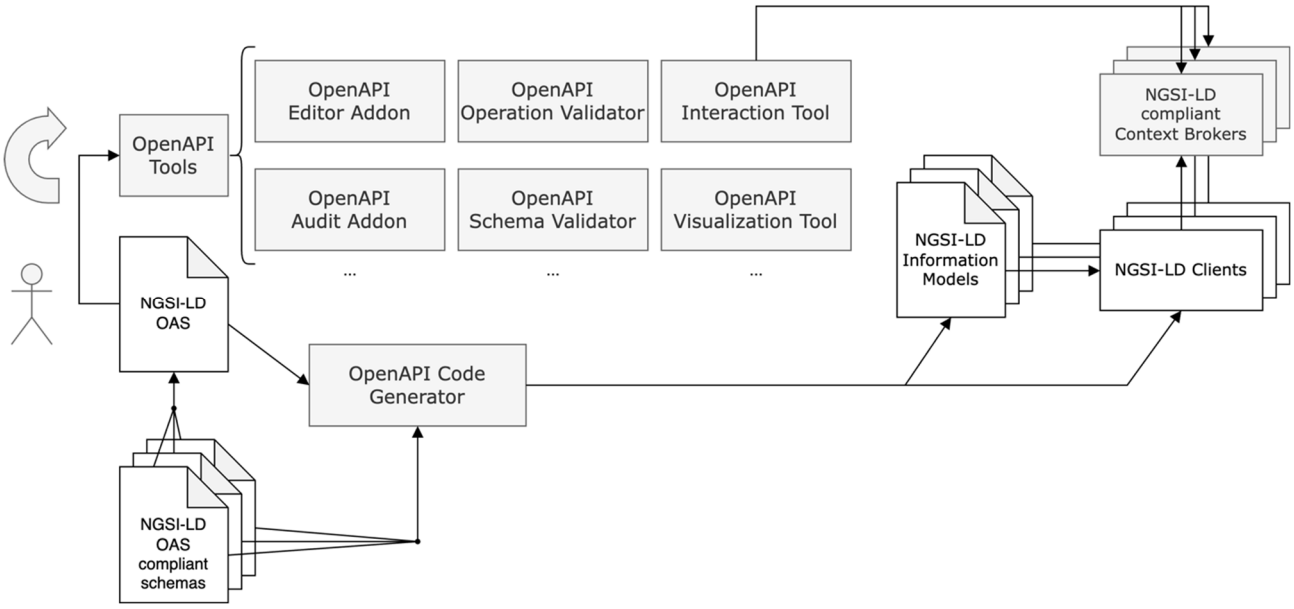

Figure 4.2.2-1: Evaluation workflow for the NGSI-LD OAS

4.2.3 Organization and description of the OpenAPI Specification

4.2.3.0 Foreword

The NGSI-LD OAS structure has been defined following good practices guidelines for OpenAPI specifications [i.3] particularized for versions 3.0.3 [4], [5] and 3.1.0 [6]. The OAS organization addresses, among other things, the supported data model or schema, API operations, as well as its description and documentation. Furthermore, it is important to include some API operation examples to help users understand how the API works. The following is a summary of the main sections of the NGSI-LD OAS:

- **openapi**: It indicates the version of the OpenAPI specification used (e.g. version 3.0.3). Using this field tools can check that the description correctly adheres to the specification.
- **info**: It provides general information about the NGSI-LD API, such as the title, version, description, license, and contact information.
- **externalDocs**: This section indicates the description and URL of the ETSI GS CIM 009 [1] specification document associated with the NGSI-LD API.
- **servers**: It provides the base URL where the NGSI-LD API is being served.
- **paths**: It describes all the endpoints of the NGSI-LD API, including their operations, different parameters, the different client-side request bodies, and all possible server-side responses. Server and client code can be generated from this description, along with its documentation.
- **components**: Often, multiple API operations have some common parameters or return the same response structure. To avoid code duplication, the common definitions can be indicated in the global "components" section and reference them. Then, the "components" section serves as a container for various reusable definitions such as schemas (i.e. data models), parameters, responses, examples, and others. As the name of the section suggests, it contains different components:
  - **headers:** It provides common headers for all API operations.
  - **parameters:** It provides common parameters for all API operations.
  - **requestBodies:** It provides the body information to be included with create and update API operations (i.e. POST, PUT, and PATCH HTTP operations). For example, when creating a resource using POST or PUT, the request body usually contains the representation of the resource to be created.
  - **schemas:** It allows the definition of the data model followed by the different data types considered within the OAS.
  - **responses:** It specifies common responses for all API operations. Each operation shall have at least one response defined, usually a successful response.

Each of the main sections within the NGSI-LD OAS structure defined above will be detailed in the following clauses.

4.2.3.1 The openapi and info sections

The "openapi" section defines the version of the OpenAPI specification used. For this sample of the NGSI-LD OAS, version 3.0.3 [4], [5] has been selected. The most significant change between OpenAPI version 3.0.3 and the latest available version, 3.1.0 [6], is that the latter allows defining JSON schemas for OpenAPI separately instead of defining the schemas in the OpenAPI specification itself. For the distribution of NGSI-LD OAS releases for interaction, development and stub code generation purposes, OpenAPI version 3.0.3 is used, since some of the tools used in the design and evaluation phases of OpenAPI are not compatible with the latest version 3.1.0 [i.4]. Instead, for visualization and documentation purposes with each NGSI-LD OAS release, OpenAPI version 3.1.0 is used. Within the "info" section, the OpenAPI specification provides additional information. First, it defines the title of the OpenAPI specification, the version of the NGSI-LD API that is covered (e.g. version 1.6.1), as well as a descriptive field. Apart from that, it specifies as a contact the URL of the ETSI CIM committee and also a license in accordance with ETSI legal matters. The following NGSI-LD OAS fragment shows the structure of these "openapi" and "info" sections for OAS version 1.6.1.

```
openapi: 3.0.3
info:
  title: NGSI-LD OAS
  version: 1.6.1
  description: NGSI-LD OpenAPI Specification.
  contact:
    url: https://www.etsi.org/committee/cim
  license:
    name: BSD-3-Clause
    url: https://forge.etsi.org/legal-matters
```

4.2.3.2 The externalDocs section

The "externalDocs" section includes a description and accessible URL of the Group Specification document from ETSI ISG CIM about the NGSI-LD API version covered by the NGSI-LD OAS. The following NGSI-LD OAS fragment shows the structure of this "externalDoc" section for OAS version 1.6.1.

```
externalDocs:
  description: ETSI GS CIM 009 V1.6.1 cross-cutting Context Information Management (CIM); NGSI-LD API
  url: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.06.01_60/gs_CIM009v010601p.pdf
```

4.2.3.3 The servers section

The "servers" section includes the URL and different variables (i.e. protocol, hostname, and port) to specify an endpoint within the NGSI-LD OAS where the NGSI-LD API is served. The following NGSI-LD OAS fragment shows the structure of this "servers" section.

```
servers:
  - url: '{protocol}://{hostname}:{port}/ngsi-ld/v1'
    variables:
      protocol:
        enum:
          - http
          - https
        default: https
      hostname:
        default: localhost
      port:
        default: '443'
```

4.2.3.4 The paths section

The "paths" section of the NGSI-OAS specifies each of the endpoints of the NGSI-LD API with their different type of operations. Each operation defines requirements as established by the NGSI-LD API. For each operation, descriptive information is initially included, adding its summarized functionality, an operation identifier, and a tag to classify the type of operation (as stated by clause 4.3.5 of ETSI GS CIM 009 [3]). In addition, the operations define their own query and request header parameters, request bodies including descriptions and particular content, and query responses including descriptions, header, and particular content. In some situations, the operations define their own request body content and their own response content, but other times they reference corresponding definitions within the "requestBody" and "responses" subsections of the NGSI-LD OAS (for more information see clause 4.2.3.5). The following NGSI-LD OAS fragment shows as an example the structure of the HTTP POST operation to create an NGSI-LD Entity.

```
/entities:
  post:
    tags:
      - Context Information Provision
    summary: |
      Entity creation
    description: |
      5.6.1 Create Entity

      This operation allows creating a new NGSI-LD Entity.
    operationId: createEntity
    parameters:
      # Local Query param
      - $ref: '#/components/parameters/Query.local'
      # Request headers
```

```
      - $ref: '#/components/parameters/Headers.Link'
      - $ref: '#/components/parameters/Headers.ngsildTenant'
    requestBody:
      description: |
        Payload body in the request contains a JSON-LD object which represents the entity that is to be
        created.
      content:
        application/json:
          schema:
            allOf:
              - $ref: '#/components/schemas/Entity'
              - required:
                - id
                - type
        application/json+ld:
          schema:
            allOf:
              - $ref: '#/components/schemas/Entity'
              - type: object
                properties:
                  '@context':
                    $ref:  '#/components/schemas/LdContext'
              - required:
                - id
                - type
                - '@context'
    responses:
      '201':
        description: |
          The HTTP response shall include a "Location" HTTP header that contains
          the resource URI of the created entity resource.
        headers:
          Location:
            $ref: '#/components/headers/Location'
          NGSILD-Tenant:
            $ref: '#/components/headers/NGSILD-Tenant'
      '207':
        headers:
          Location:
            $ref: '#/components/headers/Location'
          NGSILD-Tenant:
            $ref: '#/components/headers/NGSILD-Tenant'
        $ref: '#/components/responses/MultiStatus.BatchOperationResult'
      '400':
        $ref: '#/components/responses/BadRequest'
      '409':
        $ref: '#/components/responses/Conflict'
      '422':
        $ref: '#/components/responses/Unprocessable'
```

4.2.3.5 The components section

4.2.3.5.0 Foreword

The "components" section of the NGSI-LD OAS includes common definitions of different components considered within the different operations. This "components" section includes those common headers, parameters, schemas, request bodies, and responses that can be referenced by the operations defined within the NGSI-LD OAS.

4.2.3.5.1 The headers subsection

The "headers" subsection provides common headers defined for the responses considered within the operations of the NGSI-LD OAS. Each header specifies a description and a particular schema with data type and format. The following NGSI-LD OAS fragment shows as an example the structure of the *NGSILD-Tenant* response header, which specifies that responses include a string to identify the tenant to which the NGSI-LD HTTP operation is targeted.

```
NGSILD-Tenant:
  description: |
    6.3.14 Tenant specification. The tenant to which the NGSI-LD HTTP operation is targeted.
  schema:
    type: string
```

4.2.3.5.2 The parameters subsection

The "parameters" subsection provides common query, path, and header parameters considered within the operations of the NGSI-LD OAS. Mainly, each parameter specifies a common name, a particular description, the schema with data type and format, additional serialization rules by means of the "style" and "explode" keywords, and the "required" field to mark a parameter as required or not. The "in" keyword is a placeholder to indicate the type of parameter to be defined (i.e. query, path, or header parameter). The following NGSI-LD OAS fragment shows as an example the structure of different query, path, and header parameters.

```
Query.local:
  name: local
  in: query
  description: |
    6.3.18 Limiting Distributed Operations. If local=true then no Context Source Registrations shall be
    considered as matching to avoid cascading distributed operations (see clause 4.3.6.4).
  style: form
  explode: true
  schema:
    type: boolean
  required: false

Path.entityId:
  name: entityId
  in: path
  description: Id (URI) of the entity to be retrieved.
  schema:
    $ref: '#/components/schemas/Path'
  required: true

Headers.Link:
  name: Link
  in: header
  description: |
    6.3.5 JSON-LD @context resolution

    In summary, from a developer's perspective, for POST, PATCH and PUT operations,
    if MIME type is "application/ld+json", then the associated @context shall be provided
    only as part of the request payload body. Likewise, if MIME type is "application/json",
    then the associated @context shall be provided only by using the JSON-LD Link header.
    No mixes are allowed, i.e. mixing options shall result in HTTP response errors.
    Implementations should provide descriptive error messages when these situations arise.

    In contrast, GET and DELETE operations always take their input @context from the JSON-LD Link Header.
  explode: true
  schema:
    type: string
    format: uri
```

4.2.3.5.3 The requestBodies subsection

The "requestBodies" subsection provides the structure of the common request bodies for operations, with their corresponding information and resources to be used. Each request body allows to specify its schema structure depending on the MIME type (i.e. "application/json", "application/ld+json", or "application/geo+json"). In some situations, the request bodies define their own schema structure, but other times they combine it with references to schema definitions within the general "schemas" subsection of the NGSI-LD OAS. The following NGSI-LD OAS fragment shows as an example the structure of the *Subscription* request body. Depending on the MIME type, the *Subscription* request body define the schema structure, combining references to schemas defined within the "schemas" subsection, such as *Subscription* and *LdContext*, with additional controls of required parameters specified along the referenced schemas. For more information about the definition of schemas see clause 4.2.3.5.4.

```
Subscription:
  content:
    application/json:
      schema:
        allOf:
          - $ref: '#/components/schemas/Subscription'
          - required:
            - type
            - notifications
    application/json+ld:
      schema:
        allOf:
          - $ref: '#/components/schemas/Subscription'
          - type: object
```

```
            properties:
              '@context':
                $ref:  '#/components/schemas/LdContext'
          - required:
            - type
            - notifications
            - '@context'
```

4.2.3.5.4 The schemas subsection

The "schemas" subsection provides the common data models followed by the different data types considered within the NGSI-LD OAS. Each schema defines the data model properties in terms of data type, data format, and description. The properties can also specify additional controls such as default values, regular expressions, or maximum and minimum values. In addition, properties can reference another schemas already defined within the "schemas" subsection of the NGSI-LD OAS to specify their own characteristics. The following NGSI-LD OAS fragment shows as an example the structure of the *Entity* schema. It specifies the data model of an NGSI-LD Entity concept, including the different properties with their own particularities. In this case, the *Entity* schema includes the "additionalProperties" field for adding schemas defined for the different NGSI-LD Attributes (i.e. NGSI-LD Property and NGSI-LD Relationship concepts) considered within a NGSI-LD Entity. The "oneOf" keyword is used to specify that each additional property shall match exactly one of the defined subschemas.

```
Entity:
  description: |
    5.2.4 NGSI-LD Entity.
  type: object
  properties:
    id:
      description: |
        Entity id.
      type: string
      format: uri
    type:
      description: |
        Entity Type(s). Both short hand string(s) (type name) or URI(s) are allowed.
      oneOf:
        - type: string
        - type: array
          items:
            type: string
    scope:
      description: |
        Scope.
      oneOf:
        - type: string
        - type: array
          items:
            type: string
    location:
      description: |
        Default geospatial Property of an entity. See clause 4.7.
      $ref: '#/components/schemas/GeoProperty'
    observationSpace:
      $ref: '#/components/schemas/GeoProperty'
    operationSpace:
      $ref: '#/components/schemas/GeoProperty'
    # Clause 5.2.2 Common members. System-generated
    createdAt:
      $ref: '#/components/schemas/CreatedAt'
      readOnly: true
    modifiedAt:
      $ref: '#/components/schemas/ModifiedAt'
      readOnly: true
    deletedAt:
      $ref: '#/components/schemas/DeletedAt'
      readOnly: true
  additionalProperties:
    oneOf:
      - $ref: '#/components/schemas/Property'
      - type: array
        items:
          $ref: '#/components/schemas/Property'
      - $ref: '#/components/schemas/Relationship'
      - type: array
        items:
          $ref: '#/components/schemas/Relationship'
      - $ref: '#/components/schemas/GeoProperty'
      - type: array
```

```
        items:
          $ref: '#/components/schemas/GeoProperty'
      - $ref: '#/components/schemas/LanguageProperty'
      - type: array
        items:
          $ref: '#/components/schemas/LanguageProperty'
```

4.2.3.5.5 The responses subsection

The "responses" subsection provides the definition of the structure of the common responses considered within the NGSI-LD OAS. Each response allows to specify a description, the specific headers, as well as additional content with its schema structure depending on the MIME type. For the declaration of headers, each response header references to the related one already defined within the general "headers" subsection of the NGSI-LD OAS. In a similar way, the schemas defined within the content of the responses refer to schemas already defined within the general "schemas" subsection of the NGSI-LD OAS. The following NGSI-LD OAS fragment shows as an example the structure of the *BadRequest* response. For each MIME type, the *BadRequest* response references a *ProblemDetails* schema that provides additional error details as payload of the operation response in accordance with IETF RFC 7807 [10]. In addition, this response includes a reference to the *NGSILD-Tenant* and *NGSILD-Warning* headers. On the one hand, the *NGSILD-Tenant* header is used to optionally specify in the response the tenant to which the NGSI-LD HTTP operation was targeted, if it was previously specified in the regarding operation request. On the other hand, the *NGSILD-Warning* header is used to indicate abnormal behavior for distributed HTTP GET operations performed over the resources /entities and /entities/{entity-id}.

```
BadRequest:
  description: |
    It is used to indicate that the request or its content is incorrect,
    see clause 6.3.2. In the returned ProblemDetails structure, the "detail"
    attribute should convey more information about the error.
  headers:
    NGSILD-Tenant:
      $ref: '#/components/headers/NGSILD-Tenant'
    NGSILD-Warning:
      $ref: '#/components/headers/NGSILD-Warning'
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/ProblemDetails'
```

4.3 OpenAPI Specification repository

The NGSI-LD OAS defined by ETSI ISG CIM is available in a repository of the CIM GitLab organization within the official ETSI Forge [i.5]. In this repository different artifacts for the definition of the NGSI-LD OAS can be found.

The main normative artifacts are the NGSI-LD OAS files in YAML format for each version of the NGSI-LD API [7], [8] and [9].

Moreover, a folder containing the JSON-LD core *@context* defined by the NGSI-LD API, examples of compatible NGSI-LD payloads for multiple types of NGSI-LD operations, as well as examples for defining the OpenAPI schemas compliant with the NGSI-LD OAS can be found in the repository.

Different releases of the NGSI-LD OAS should be managed in the repository using git tags. Each tag identifies a stable version of the NGSI-LD OAS. The name of the tag matches the version of the NGSI-LD API that the OAS is implementing. Table 4.3-1 shows the correspondence between versions of the NGSI-LD API and releases of the NGSI-LD OAS, including the reference to the access URL for the NGSI-LD OAS release available within each repository tag. It should also be noted that different branches are used in the repository for the development phase of each release of the NGSI-LD OAS. Whenever a new version of the NGSI-LD API is defined, a new branch will be created to develop the corresponding OAS and therefore a new tag will be created when the specification is stable. Additionally, the main branch of the repository will match the latest available version of the NGSI-LD OAS.

Each release of NGSI-LD OAS provides an option for use in visualization tools based on OpenAPI version 3.1.0 and another option for testing and development purposes based on OpenAPI version 3.0.3. This is because most testing and development tools for OpenAPI specifications, such as interaction and stub code generation tools, are not yet supported in the latest version 3.1.0.

Table 4.3-1: NGSI-LD OAS releases

NGSI-LD API version	NGSI-LD OAS release
NGSI-LD API v1.6.1 [1]	NGSI-LD OAS v1.6.1 [7]
NGSI-LD API v1.7.1 [2]	NGSI-LD OAS v1.7.1 [8]
NGSI-LD API v1.8.1 [3]	NGSI-LD OAS v1.8.1 [9]

Annex A (informative): Recommended tools for the design and evaluation of OpenAPI Specification

A.1 Introduction

This annex is informative and is intended to present some open-source tools suggested by the OpenAPI Initiative [i.1], [i.2], that are useful to facilitate the design and evaluation of the NGSI-LD OAS.

A.2 Tools for the design of OpenAPI Specification

To facilitate the design and development of OpenAPI specifications, the OpenAPI Swagger Editor [i.7] extension in Visual Studio Code [i.6] is recommended. This extension is based on the official Swagger Editor tool, which is an open-source API editor to design, describe, and document APIs with the OpenAPI specifications [i.7]. The features of this extension include SwaggerUI [i.8] and ReDoc [i.9] preview, schema enforcement, code navigation, definition links, static security analysis, and more. SwaggerUI allows users to visualize and interact with the API's resources without having any of the implementation logic in place. ReDoc is a utility to easily generate and preview OpenAPI documentation. Annex B shows how SwaggerUI and ReDoc can be used to navigate the NGSI-LD OAS. In addition, for generating the documentation of the OpenAPI, there are alternative solutions such as Scalar [i.10] and OpenDocumenter [i.11].

A.3 Tools for the evaluation of OpenAPI Specification

For the evaluation of OpenAPI specifications, different utilities are recommended by the OpenAPI Initiative [i.1]. The OpenAPI Swagger Editor [i.7] extension, which is aforementioned in clause A.2, integrates an API Contract Security Audit [i.12] tool to identify and fix issues, as well as to check the quality of the OAS. In addition, for extra validation of the schemas and operations within the OpenAPI, there are different tools such as Swagger 2.0 and OpenAPI 3.0 parser/validator [i.13], and Express OpenAPI Validator [i.14].

Annex B (informative): OpenAPI visualization and interaction tools

B.1 Introduction

This annex is informative and is intended to show how the SwaggerUI [i.8] and ReDoc [i.9] tools can be used to navigate the NGSI-LD OAS for interaction and visualization purposes.

B.2 SwaggerUI

SwaggerUI [i.8] allows users to visualize and interact with the API's resources without having any of the implementation logic in place. It is automatically generated from an OAS, with a visual documentation making it easy for backend implementation and client-side consumption. Figure B.2-1 represents, as an example, a simple snapshot of the GUI offered by SwaggerUI for the release 1.7.1 of NGSI-LD OAS [i.15]. SwaggerUI currently only supports OpenAPI version 3.0.3, but it may be the most convenient solution to learn about the main options of the different operations defined within the NGSI-LD OAS and to be able to interact directly with the API of an NGSI-LD server.

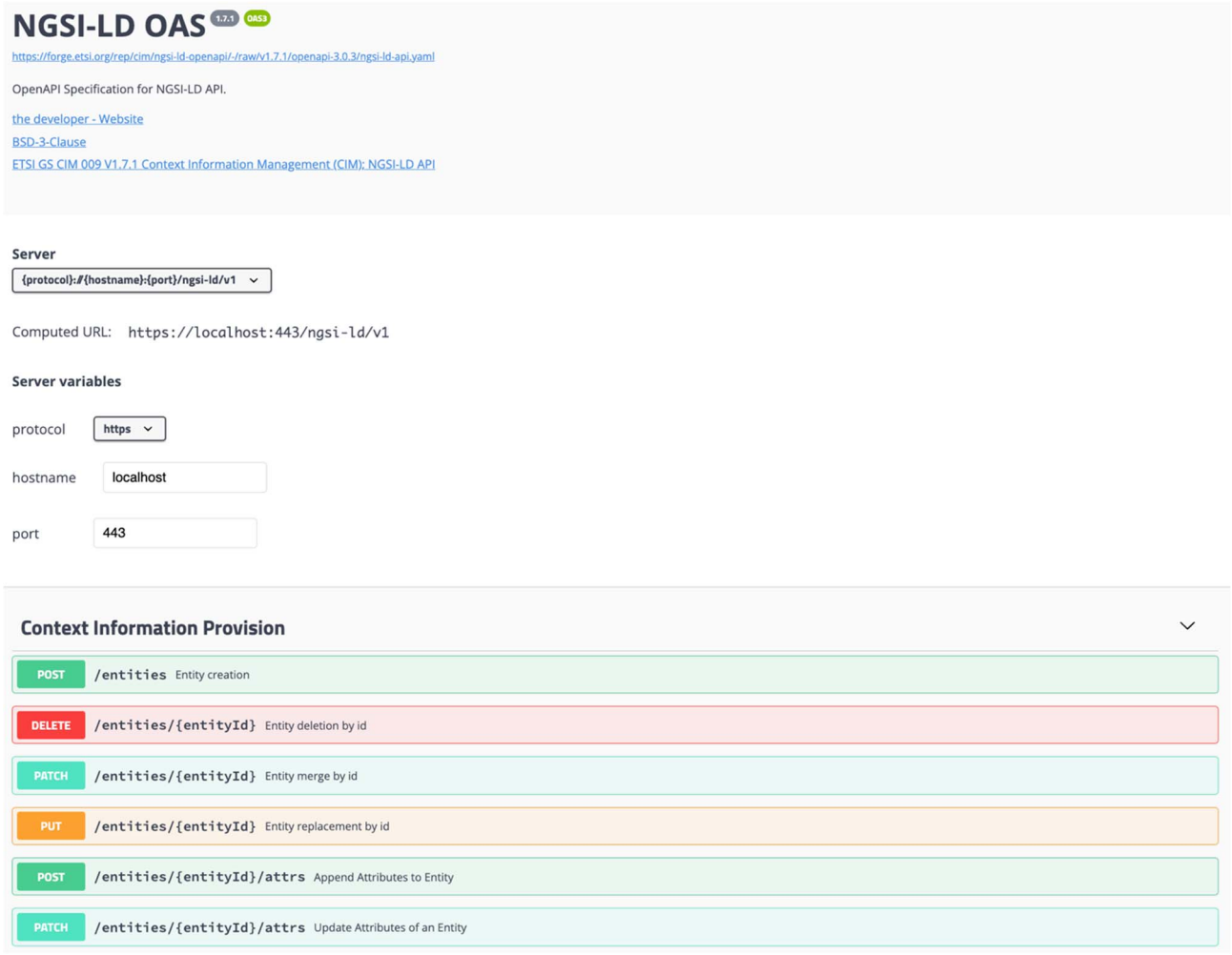

Figure B.2-1: SwaggerUI [i.8] for the NGSI-LD OAS

B.3 ReDoc

ReDoc [i.9] is an alternative open-source tool for generating documentation from OpenAPI specifications. Basically, ReDoc API offers a GUI with a navigation menu to facilitate both the documentation and examples of requests and responses for operations considered within an OAS. Figure B.3-1 represents, as an example, a simple snapshot of the GUI offered by ReDoc for the release 1.7.1 of NGSI-LD OAS [i.16]. ReDoc supports OpenAPI version 3.1.0 and is the most convenient and recommended tool for viewing the NGSI-LD OAS documentation in a more readable way.

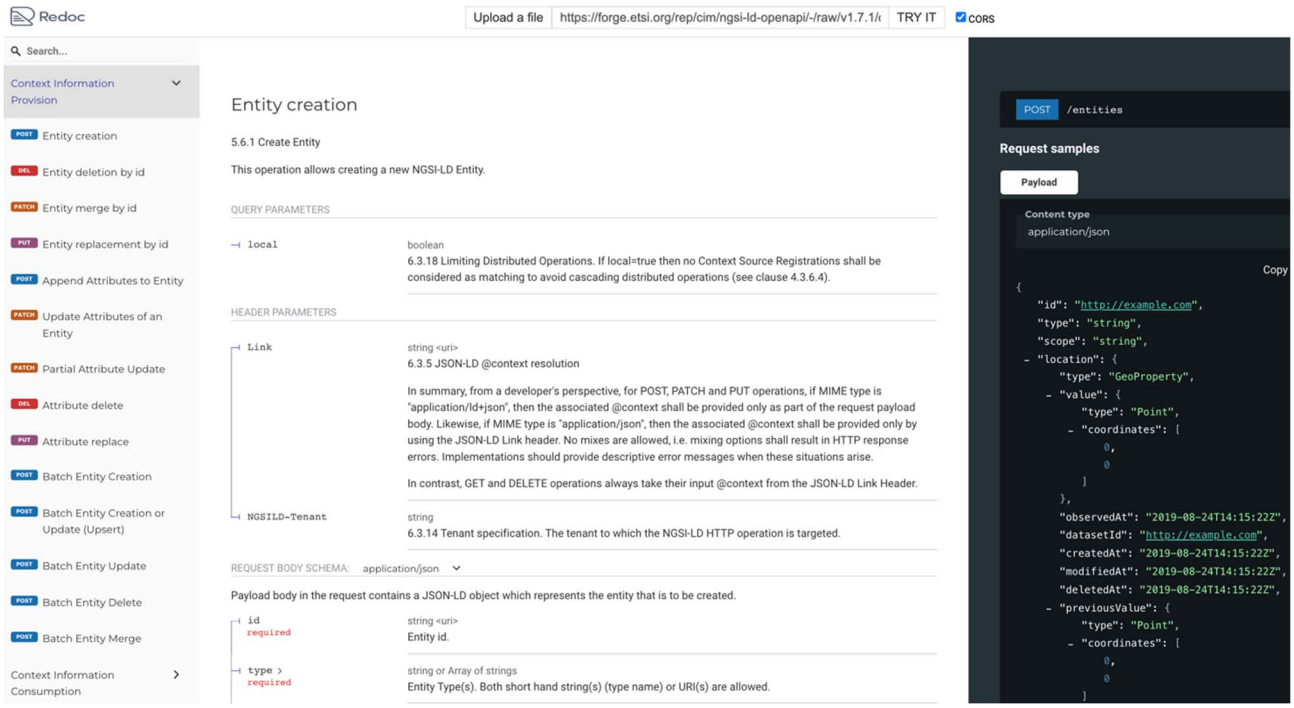

Figure B.3-1: ReDoc [i.9] for the NGSI-LD OAS

Annex C (informative): Guidelines for defining custom schemas compliant with the OpenAPI Specification

C.1 Introduction

This annex is informative and is intended to show how to define custom OpenAPI schemas that are compatible with the NGSI-LD OAS. It allows developers to define specific-purpose applications where they model their own OpenAPI schemas, so they are compatible with the NGSI-LD meta-model defined within the NGSI-LD OAS. In this way, if the user has an NGSI-LD API client in a particular programming language, the defined custom schemas facilitate the availability of the resulting NGSI-LD information models as programming code to be used within the NGSI-LD API client and instantiate it accordingly in a corresponding Context Broker. The schemas represented in this annex are exemplified for a vehicular use case.

C.2 Example of OpenAPI schemas for vehicular use case

The following examples show customizable OpenAPI schemas compliant with the NGSI-LD OAS metamodel schemas for a use case to model information about vehicles by following the sample information model proposed in ETSI GS CIM 009 [3].

Figure C.2-1 depicts the high-level representation using a UML diagram of the proposed NGSI-LD information model for the vehicular use case according to the OpenAPI schemas specified below. The "*" character represents required NGSI-LD Properties of the NGSI-LD Entities. There are different types of NGSI-LD Relationships that have one-to-one cardinality *(0..1)*, since they represent relationships with only one possible instance of the target NGSI-LD Entity. Meanwhile, there are NGSI-LD Relationships that have one-to-many cardinality *(0..N)*, as they represent relationships with one or more possible instances of the target NGSI-LD Entity. All these NGSI-LD information model conventions are included in the OpenAPI schemas as mentioned above.

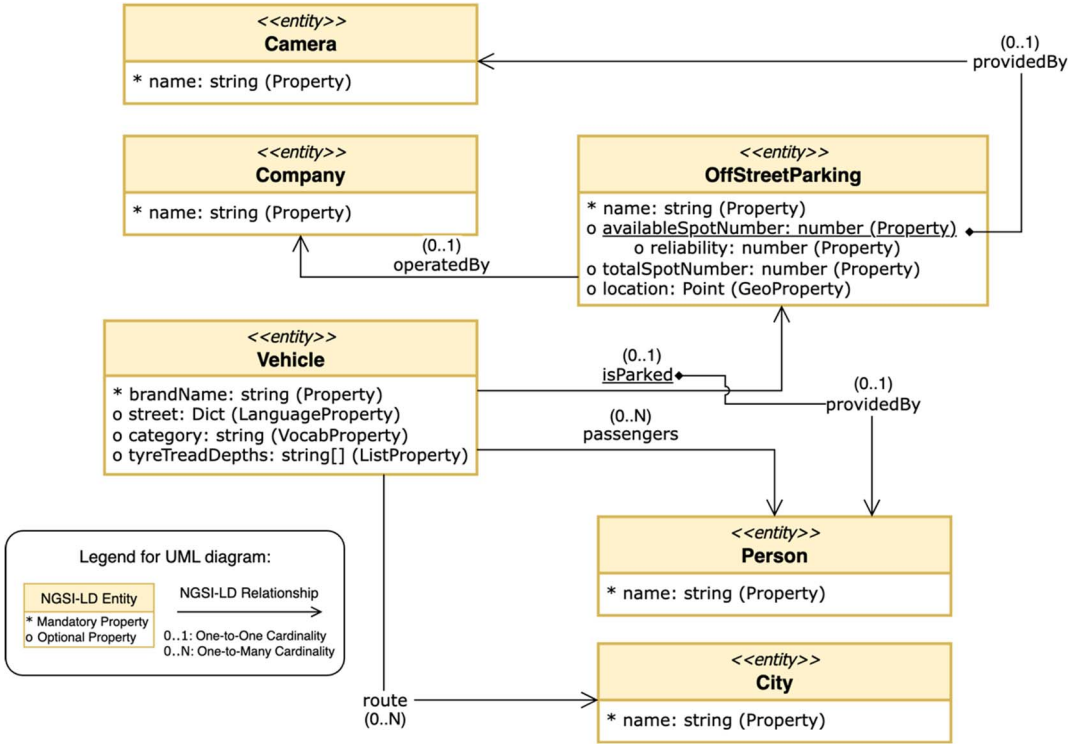

Figure C.2-1: UML diagram about a high-level representation of an NGSI-LD information model for vehicular use case

The first example shows custom OpenAPI schemas defined within a YAML file for an NGSI-LD Entity named *Vehicle* and its different attributes (i.e. *BrandName*, *Street*, *IsParked*, *Category*, *TyreTreadDepths*, *Passengers*, and *Route*) which are compliant with the different related schemas defined within the NGSI-LD OAS. Each custom schema references the NGSI-LD OAS base schema (i.e. *Entity*, *Property*, *Relationship*, *ListProperty*, *ListRelationship*, *LanguageProperty*, *VocabProperty*, and *GeoProperty*) and incorporates new properties in addition to descriptive information. The *Vehicle* schema adds that the Entity type field has to be *Vehicle* and that it has to include different properties that reference the schemas also defined, indicating which fields are required for the construction of the NGSI-LD Entity of type *Vehicle*.

Between the schemas of the attributes for *Vehicle* there is *IsParked* derived for an NGSI-LD Relationship to point out the object of an NGSI-LD Entity of type *OffStreetParking* in order to indicate the parking where is the vehicle. This *IsParked* schema also references another schema named *ProvidedBy* to specify an additional attribute which is an NGSI-LD Relationship to point out the object of an NGSI-LD Entity of type *Person* in order to indicate the person who provides the parking spot. Additional schemas such as *Passengers* and *Route* exist to define custom NGSI-LD Relationships to indicate the list of persons who are passengers of the vehicle and the list of cities which are covered on the vehicle's route. This type of schemas dedicated to NGSI-LD Relationships add as a required property an object field typically considered in an NGSI-LD Relationship that has to be of type string in order to indicate unique Entity identifiers of the target NGSI-LD Entities.

Additionally, there are custom schemas for other *Vehicle* attributes such as *BrandName*, *Street*, *Category*, and *TyreTreadDepths*, which reference schemas for NGSI-LD Property, NGSI-LD LanguageProperty, NGSI-LD VocabProperty, and NGSI-LD ListProperty relatively. Each of these custom schemas adds as required property the data type and its reserved name depending on the NGSI-LD OAS base schema it references.

```
openapi: 3.0.3
info:
  title: Example schemas for vehicle information
  version: 0.0.1
  description: |
    Example schemas compliant with the NGSI-LD OAS metamodel according to ETSI GS CIM 009.
paths: {}
components:
  schemas:
    Vehicle:
      description: |
        NGSI-LD Entity Type that represents a vehicle.
      allOf:
        - $ref: https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Entity'
        - type: object
          properties:
            type:
              description: NGSI-LD Entity identifier. It has to be Vehicle.
              type: string
              enum:
                - Vehicle
              default: Vehicle
            brandName:
              $ref: '#/components/schemas/BrandName'
            street:
              $ref: '#/components/schemas/Street'
            isParked:
              $ref: '#/components/schemas/IsParked'
            category:
              $ref: '#/components/schemas/Category'
            tyreTreadDepths:
              $ref: '#/components/schemas/TyreTreadDepths'
            passengers:
              $ref: '#/components/schemas/Passengers'
            route:
              $ref: '#/components/schemas/Route'
        - required:
            - type
            - brandName
    BrandName:
      description: |
        NGSI-LD Property Type. The vehicle brand name.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Property'
        - type: object
          properties:
            value:
              type: string
          required:
            - value
      additionalProperties: false
    Street:
      description: |
        NGSI-LD LanguageProperty Type. The vehicle street.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
                 #/components/schemas/LanguageProperty'
        - type: object
          properties:
            languageMap:
```

```
            type: object
          required:
            - languageMap
    additionalProperties: false
  IsParked:
    additionalProperties: false
    description: |
      NGSI-LD Relationship type to identify the parking where is the vehicle (i.e. the identifier of an NGSI-LD Entity
      of type OffStreetParking).
    allOf:
      - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
               #/components/schemas/Relationship'
      - type: object
        properties:
          object:
            type: string
            format: uri
          objectType:
            type: string
            format: uri
          providedBy:
            $ref: '#/components/schemas/ProvidedBy'
        required:
          - object
  Category:
    description: |
      NGSI-LD VocabProperty Type. The vehicle category.
    allOf:
      - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
               #/components/schemas/VocabProperty'
      - type: object
        properties:
          vocab:
            type: string
        required:
          - vocab
    additionalProperties: false
  TyreTreadDepths:
    description: |
      NGSI-LD ListProperty Type. The vehicle tyre tread depths.
    allOf:
      - $ref: ''https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
               #/components/schemas/ListProperty'
      - type: object
        properties:
          valueList:
            items:
              type: string
        required:
          - valueList
    additionalProperties: false
  Passengers:
    description: |
      NGSI-LD Relationship type to identify the passengers of the vehicle (i.e. the identifier of an NGSI-LD Entity of
      type Person).
    allOf:
      - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
               #/components/schemas/Relationship'
      - type: object
        properties:
          object:
            type: array
            items:
              type: string
              format: uri
          objectType:
            type: string
            format: uri
        required:
          - object
    additionalProperties: false
  Route:
    description: |
      NGSI-LD ListRelationship type to identify the route of the vehicle (i.e. the list of identifiers of NGSI-LD
      Entities of type City).
    allOf:
      - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
               #/components/schemas/ListRelationship'
      - type: object
        properties:
          objectList:
            type: array
            items:
              type: object
              format: uri
          objectType:
            type: string
            format: uri
        required:
          - objectList
    additionalProperties: false
  ...
```

Similarly, a second example shows in another fragment of the previous YAML file other custom OpenAPI schemas for an NGSI-LD Entity named *OffStreetParking* and its attributes (i.e. *AvailableSpotNumber, TotalSpotNumber*, and *OperatedBy*) which are compliant with the *Entity*, *Property*, and *Relationship* related schemas defined within the NGSI-LD OAS. Again, each custom schema references the NGSI-LD OAS base schema and incorporates additional information. The *OffStreetParking* schema adds that the Entity type has to be *OffStreetParking* and includes different properties that reference the related schemas of its attributes, indicating which fields are required. The *AvailableSpotNumber* and *TotalSpotNumber* schemas represent NGSI-LD Properties of the NGSI-LD Entity of type *OffStreetParking*, adding that the value property has to be of type number. The *AvailableSpotNumber* schema also includes a reference to the *Reliability* schema to add an additional NGSI-LD Property of type number, and a reference to the *ProvidedBy* schema to specify an additional NGSI-LD Relationship to point to the object of an NGSI-LD Entity of type *Camera* to indicate the camera provided in the parking spot number. In addition, the *OperatedBy* schema represents an NGSI-LD Relationship of the NGSI-LD Entity of type *OffStreetParking* to point out the object of an NGSI-LD Entity of type *Company* to indicate the parking company.

```
components:
  schemas:
    ...
    OffStreetParking:
      description: |
        NGSI-LD Entity Type that represents a parking.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
                 #/components/schemas/Entity'
        - type: object
          properties:
            type:
              description: NGSI-LD Entity identifier. It has to be OffStreetParking.
              type: string
              enum:
                - OffStreetParking
              default: OffStreetParking
            name:
              $ref: '#/components/schemas/Name'
            availableSpotNumber:
              $ref: '#/components/schemas/AvailableSpotNumber'
            totalSpotNumber:
              $ref: '#/components/schemas/TotalSpotNumber'
            operatedBy:
              $ref: '#/components/schemas/OperatedBy'
        - required:
            - type
            - name
    AvailableSpotNumber:
      description: |
        NGSI-LD Property Type. The available spot number within a parking.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Property'
        - type: object
          properties:
            value:
              type: number
            reliability:
              $ref: '#/components/schemas/Reliability'
            providedBy:
              $ref: '#/components/schemas/ProvidedBy'
          required:
            - value
      additionalProperties: false
    Reliability:
      description: |
        NGSI-LD Property Type. The reliability of the available spot number within a parking.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Property'
        - type: object
          properties:
            value:
              type: number
          required:
            - value
      additionalProperties: false
    TotalSpotNumber:
      description: |
        NGSI-LD Property Type. The total spot number within a parking.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas Property'
        - type: object
          properties:
            value:
              type: number
          required:
            - value
      additionalProperties: false
    OperatedBy:
      description: |
        NGSI-LD Relationship type to identify the company that operates the parking (i.e. the identifier of an NGSI-LD
        Entity of type Company).
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
                 #/components/schemas/Relationship'
```

```
        - type: object
          properties:
            object:
              type: string
          required:
            - object
      additionalProperties: false
    ...
```

Similarly, a third example shows in another fragment of the YAML file other custom OpenAPI schemas for NGSI-LD Entities named *Person*, *City*, *Camera*, and *Company*. These NGSI-LD Entities only incorporate references to a schema *Name* which defines an NGSI-LD Property to provide a name for them.

```
components:
  schemas:
    ...
    Person:
      description: |
        NGSI-LD Entity Type that represents a person.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Entity'
        - type: object
          properties:
            type:
              description: NGSI-LD Entity identifier. It has to be Person.
              type: string
              enum:
                - Person
              default: Person
            name:
              $ref: '#/components/schemas/Name'
        - required:
            - type
            - name
    City:
      description: |
        NGSI-LD Entity Type that represents a city.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Entity'
        - type: object
          properties:
            type:
              description: NGSI-LD Entity identifier. It has to be City.
              type: string
              enum:
                - City
              default: City
            name:
              $ref: '#/components/schemas/Name'
        - required:
            - type
            - name
    Camera:
      description: |
        NGSI-LD Entity Type that represents a camera.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Entity'
        - type: object
          properties:
            type:
              description: NGSI-LD Entity identifier. It has to be Camera.
              type: string
              enum:
                - Camera
              default: Camera
            name:
              $ref: '#/components/schemas/Name'
        - required:
            - type
            - name
    Company:
      description: |
        NGSI-LD Entity Type that represents a company.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Entity'
        - type: object
          properties:
            type:
              description: NGSI-LD Entity identifier. It has to be Company.
              type: string
              enum:
                - Company
              default: Company
            name:
              $ref: '#/components/schemas/Name'
        - required:
            - type
            - name
    ...
```

Finally, a last example shows in another fragment of the same YAML file other custom OpenAPI schemas for an NGSI-LD Property named *Name* and an NGSI-LD Relationship named *ProvidedBy* which are used by other customize schemas aforementioned above.

```
components:
  schemas:
    ...
    Name:
      description: |
        NGSI-LD Property Type. The natural name of an entity.
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml#/components/schemas/Property'
        - type: object
          properties:
            value:
              type: string
          required:
            - value
      additionalProperties: false
    ProvidedBy:
      additionalProperties: false
      description: |
        NGSI-LD Relationship type to identify the entity that provides something (i.e. the identifier of an NGSI-LD Entity
        of particular type).
      allOf:
        - $ref: 'https://forge.etsi.org/rep/cim/ngsi-ld-openapi/-/raw/master/ngsi-ld-api.yaml
                 #/components/schemas/Relationship'
        - type: object
          properties:
            object:
              type: string
              format: uri
          required:
            - object
    ...
```

Annex D (informative): Stub code generation and examples of use

D.1 Introduction

This annex is informative and is intended to show the utility of the NGSI-LD OAS to generate stub code for different programming languages. Examples are provided for generating and using the code for NGSI-LD API clients.

D.2 Example of Python-based NGSI-LD client

This clause shows how to use a Python-based NGSI-LD API client-side library generated from NGSI-LD OAS by using the OpenAPI Generator [i.17] tool from OpenAPI Initiative. [i.1]

Figure D.2-1 depicts in a diagram the instantiation of a resulting NGSI-LD information model for the vehicular use case by following the sample information model proposed in ETSI GS CIM 009 [3]. For each NGSI-LD Entity Type represented, there is an identifier specified in the form of a URN. The NGSI-LD unit code for representing the value of vehicle tyre tread depths in millimeters is extracted from UNECE/CEFACT Common Codes [11] for defining the units of measurement as specified by ETSI GS CIM 009 [3].

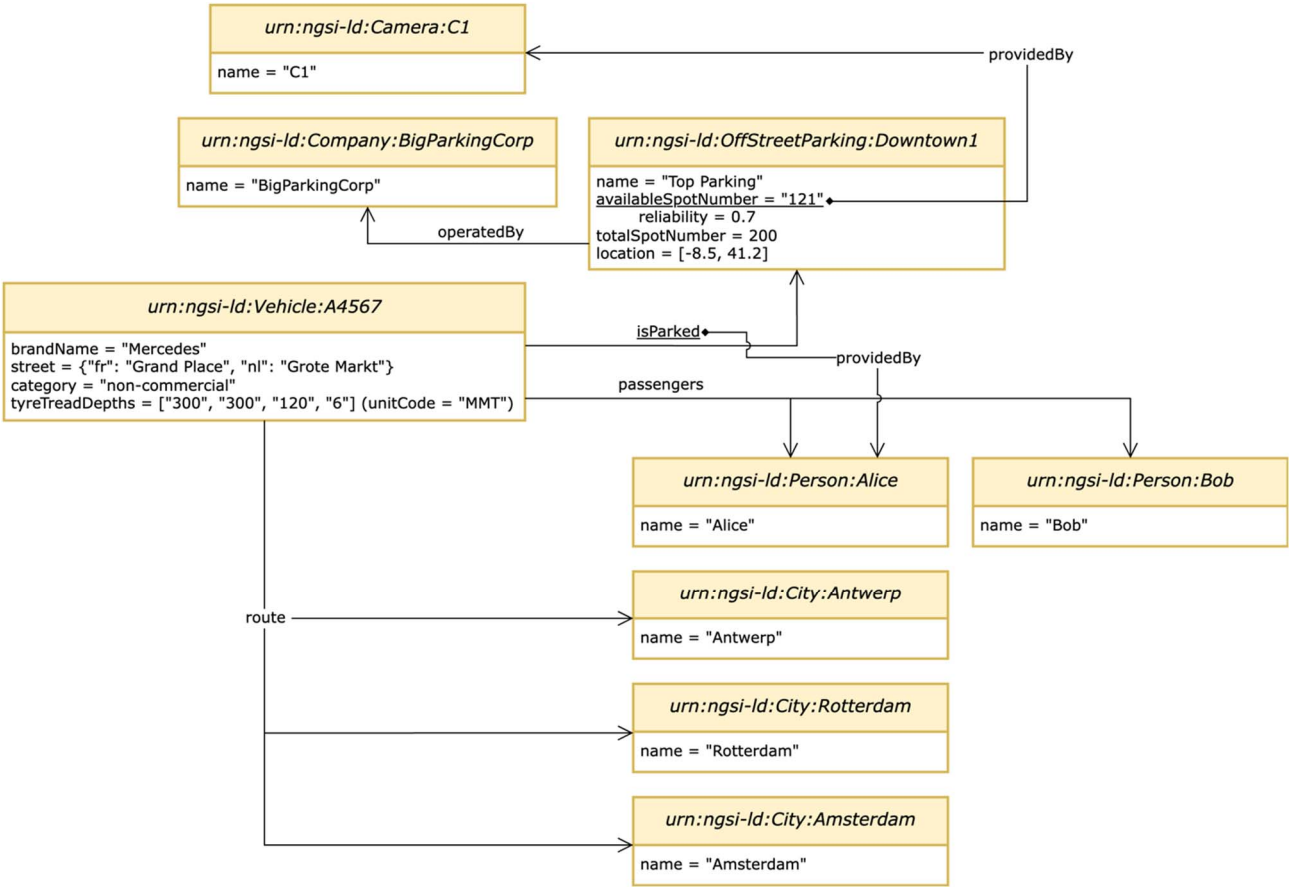

Figure D.2-1: Diagram about the instantiation representation of a sample NGSI-LD information model for vehicular use case

Below is an example of a Python code snippet that uses a generated Python-based NGSI-LD API client library for creating samples of NGSI-LD Entities of type *Vehicle*, *OffStreetParking*, *City*, *Person*, *Company*, and *Camera* using custom OpenAPI schemas previously defined and considered in Annex C. Using these custom schemas, the OpenAPI Generator [i.17] tool generates programming code in form of Python classes that could be used with the NGSI-LD API client in order to build Python objects and instantiate the resulting NGSI-LD information models. In the following two examples, both an implementation of the NGSI-LD context broker and a service that stores the NGSI-LD *@context* vocabulary in a catalogue are considered to be available and locally accessible.

```
import yaml
import os
import ngsi_ld_client

# Importing Python library modules to use the OpenAPI schemas defined for the vehicular use case:
from ngsi_ld_models.models.vehicle import Vehicle
from ngsi_ld_models.models.off_street_parking import OffStreetParking
from ngsi_ld_models.models.available_spot_number import AvailableSpotNumber
from ngsi_ld_models.models.person import Person
from ngsi_ld_models.models.camera import Camera
from ngsi_ld_models.models.company import Company
from ngsi_ld_models.models.city import City
from ngsi_ld_models.models.is_parked import IsParked
from ngsi_ld_models.models.passengers import Passengers
from ngsi_ld_models.models.route import Route
from ngsi_ld_models.models.provided_by import ProvidedBy
from ngsi_ld_models.models.operated_by import OperatedBy

# Importing Python library modules to use the self-defined OpenAPI schemas within the NGSI-LD OAS:
from ngsi_ld_client.models.entity import Entity
from ngsi_ld_models.models.geo_property import GeoProperty
from ngsi_ld_models.models.geometry_point import GeometryPoint
from ngsi_ld_models.models.geometry import Geometry
from ngsi_ld_client.models.query_entity200_response_inner import QueryEntity200ResponseInner

# Importing Python library modules to use the NGSI-LD API client:
from ngsi_ld_client.api_client import ApiClient as NGSILDClient
from ngsi_ld_client.configuration import Configuration as NGSILDConfiguration
from ngsi_ld_client.exceptions import ApiException

import time
import numpy as np

# NGSI-LD Context Broker
BROKER_URI = os.getenv("BROKER_URI", "http://localhost:9090/ngsi-ld/v1")

# Context catalogue service
CONTEXT_CATALOGUE_URI = os.getenv("CONTEXT_CATALOGUE_URI", "http://context-catalogue:8080/context.jsonld")

# Init NGSI-LD Client
configuration = NGSILDConfiguration(host=BROKER_URI)
configuration.debug = True
ngsi_ld = NGSILDClient(configuration=configuration)

ngsi_ld.set_default_header(
    header_name="Link",
    header_value='<{0}>; '
                 'rel="http://www.w3.org/ns/json-ld#context"; '
                 'type="application/ld+json"'.format(CONTEXT_CATALOGUE_URI)
)

ngsi_ld.set_default_header(
    header_name="Accept",
    header_value="application/json"
)

# Declaring API for Context Information Provision operations:
api_instance = ngsi_ld_client.ContextInformationProvisionApi(ngsi_ld)

parking_company = Company(
    id="urn:ngsi-ld:Company:BigParkingCorp",
    type="Company",
    name={"type":"Property", "value": "BigParkingCorp"}
)

entity_input = parking_company.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type Company: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
```

```
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

parking_camera = Camera(
    id="urn:ngsi-ld:Camera:C1",
    type="Camera",
    name={"type":"Property", "value": "C1"}
)

entity_input = parking_camera.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type Camera: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

availableSpotNumber=AvailableSpotNumber(
    observed_at=observed_at,
    value=121,
    reliability={"type":"Property", "value":0.7},
    providedBy=ProvidedBy.from_dict({"type": "Relationship", "object": "urn:ngsi-ld:Camera:C1"})
)

parking_location = GeometryPoint(
    type="Point",
    coordinates=[-8.5, 41.2]
)

parking_location=Geometry.from_dict(parking_location.to_dict())

geoproperty_location = GeoProperty(
    type="GeoProperty",
    value=parking_location
)

parking=OffStreetParking(
    id="urn:ngsi-ld:OffStreetParking:Downtown1",
    type="OffStreetParking",
    location=GeoProperty.from_dict({"type":"GeoProperty", "value":parking_location.to_dict()}),
    name={"type":"Property", "value":"Top Parking"},
    operatedBy=OperatedBy.from_dict({"type":"Relationship", "object": "urn:ngsi-
               ld:Company:BigParkingCorp"}),
    availableSpotNumber=availableSpotNumber,
    totalSpotNumber={"type":"Property", "value": 200}
)

entity_input = parking.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type OffStreetParking: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

person_bob=Person(
    id="urn:ngsi-ld:Person:Bob",
    type="Person",
    name={"type":"Property", "value": "Bob"}
)

entity_input = person_bob.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type Person: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

person_alice=Person(
    id="urn:ngsi-ld:Person:Bob",
    type="Person",
    name={"type":"Property", "value": "Alice"}
)

entity_input = person_alice.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)
```

```
try:
    # Create NGSI-LD entity of type Person: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

city_antwerp=City(
    id="urn:ngsi-ld:City:Antwerp",
    type="City",
    name={"type":"Property", "value": "Antwerp"}
)

entity_input = city_antwerp.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type City: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)


city_rotterdam=City(
    id="urn:ngsi-ld:City:Rotterdam",
    type="City",
    name={"type":"Property", "value": "Rotterdam"}
)

entity_input = city_rotterdam.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type City: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

city_amsterdam=City(
    id="urn:ngsi-ld:City:Amsterdam",
    type="City",
    name={"type":"Property", "value": "Amsterdam"}
)

entity_input = city_amsterdam.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type City: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)

isParked = IsParked(
    type="Relationship",
    object="urn:ngsi-ld:OffStreetParking:Downtown1",
    objectType="OffStreetParking",
    providedBy=ProvidedBy.from_dict({"type": "Relationship", "object": "urn:ngsi-ld:Person:Bob"}),
)

passengers = Passengers(
    type="Relationship",
    object=["urn:ngsi-ld:Person:Alice", "urn:ngsi-ld:Person:Bob"],
    objectType="Person",
)

route = Route(
    type="ListRelationship",
    objectList=[{"object": "urn:ngsi-ld:City:Antwerp"}, {"object": "urn:ngsi-ld:City:Rotterdam"},
                {"object": "urn:ngsi-ld:City:Amsterdam"}],
    objectType="City"
)

vehicle = Vehicle(
    id="urn:ngsi-ld:Vehicle:A4567",
    type="Vehicle",
    brandName={"type":"Property", "value":"Mercedes"},
    street={"type":"LanguageProperty", "languageMap": {"fr": "Grand Place", "nl": "Grote Markt"}},
    isParked=isParked,
    category={"type":"VocabProperty", "vocab": "non-commercial"},
```

```
    tyreTreadDepths={"type":"ListProperty", "valueList": ["300", "300", "120", "6"], "unitCode": "MMT"},
    passengers=passengers,
    route=route
)

entity_input = vehicle.to_dict()

query_entity_input = QueryEntity200ResponseInner.from_dict(entity_input)

try:
    # Create NGSI-LD entity of type Vehicle: POST /entities
    api_instance.create_entity(query_entity200_response_inner=query_entity_input)
except Exception as e:
    print("Exception when calling ContextInformationProvisionApi->create_entity: %s\n" % e)
```

Below is another example of a Python code snippet that uses a generated Python-based NGSI-LD API client library for querying NGSI-LD Entities of type *Vehicle* and *OffStreetParking*.

```
import os
import ngsi_ld_client

from ngsi_ld_client.api_client import ApiClient as NGSILDClient
from ngsi_ld_client.configuration import Configuration as NGSILDConfiguration
from ngsi_ld_client.exceptions import ApiException

# NGSI-LD Context Broker
BROKER_URI = os.getenv("BROKER_URI", "http://localhost:9090/ngsi-ld/v1")

# Context catalogue service
CONTEXT_CATALOGUE_URI = os.getenv("CONTEXT_CATALOGUE_URI",
                                  "http://context-catalogue:8080/context.jsonld")
# Init NGSI-LD Client
configuration = NGSILDConfiguration(host=BROKER_URI)
configuration.debug = True
ngsi_ld = NGSILDClient(configuration=configuration)
ngsi_ld.set_default_header(
    header_name="Link",
    header_value='<{0}>; '
                 'rel="http://www.w3.org/ns/json-ld#context"; '
                 'type="application/ld+json"'.format(CONTEXT_CATALOGUE_URI)
)

ngsi_ld.set_default_header(
    header_name="Accept",
    header_value="application/json"
)

api_instance = ngsi_ld_client.ContextInformationConsumptionApi(ngsi_ld)

try:
    # Query NGSI-LD entities of type Vehicule: GET /entities
    api_response = api_instance.query_entity(type='Vehicle')
    vehicle_entities = api_response
    for vehicle_entity in vehicle_entities:
        print(vehicle_entity.to_dict())
except Exception as e:
    print("Exception when calling ContextInformationConsumptionApi->query_entity: %s\n" % e)

try:
    # Query NGSI-LD entities of type OffStreetParking: GET /entities
    api_response = api_instance.query_entity(type='OffStreetParking')
    parking_entities = api_response
    for parking_entity in parking_entities:
        print(parking_entity.to_dict())
except Exception as e:
    print("Exception when calling ContextInformationConsumptionApi->query_entity: %s\n" % e)
```

Annex E (informative): Bibliography

- ETSI GS CIM 008 (V1.2.1): "Context Information Management (CIM); NGSI-LD Primer".
- ETSI GS CIM 006: "Context Information Management (CIM); Information Model (MOD0)".
- Generic documentation tools, examples, @context files, API specification playground for the NGSI-LD API defined by ETSI ISG CIM.

Annex F (informative): Change history

Date	Version	Information about changes
February 27th 2024	V0.0.1	Early Draft
May 2024	V0.0.9	First Stable Draft approved by ISG CIM
May 2024	V0.0.10	Review of Stable Draft
September 2024	V0.1.0	First Final Draft
October 2024	V0.1.1	Final Draft approved by ISG CIM Technical Officer review before EditHelp publication pre-processing after TB approval
November 2024	V1.1.1 V1.1.2	Publication of V1.1.1 followed by an editorial revision V1.1.2

History

Document history		
V1.1.1	November 2024	Publication
V1.1.2	December 2024	Publication