ETSI CTI Plugtests Report V1.1.1 (2025-05)

1st NGSI-LD Plugtests
Sophia Antipolis, France
24 February – 28 February 2025

PLUGTESTS™
INTEROP EVENTS

Co-funded by
the European Union

Co-funded by
EFTA

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.

1 Executive summary

The NGSI-LD Plugtests event, organized by ETSI, was held in Sophia Antipolis, France from 24th of February to 28th of February 2025. This initiative represented an important opportunity for validation and collaboration among various stakeholders involved in the development and implementation of the NGSI-LD standard, aimed at promoting interoperability of context-aware and semantic data systems. Throughout the event, intensive testing activities were carried out on different broker platforms to evaluate their compliance with the standard and their ability to interoperate in distributed scenarios.

The main goal of the Plugtest was twofold: on one hand, to validate the implementers' adherence to the NGSI-LD standard in terms of technical compliance; on the other, to test interoperability among different implementations in real-world contexts. The event also aimed to identify bugs and ambiguities in the NGSI-LD specifications, collecting feedback for future improvements. By simulating use-case scenarios and adopting automated testing tools (such as ETSI's Robot Framework, Kereval's ITB platform, and IUDX test suites), it was possible to rigorously test the brokers through complex, practical test cases.

Numerous stakeholders from the NGSI-LD ecosystem took part in the event, including developers and representatives from open-source projects and institutions. The brokers tested included City Data Hub, Fiware Lepus, Orion-LD, Scorpio, Stellio, and the Fraunhofer IIS broker. These participants actively contributed to testing sessions and technical discussions, highlighting implementation issues and proposing solutions. Members from the ISG CIM group and representatives from the IUDX project also joined the event, working together to align their respective frameworks and test suites with the NGSI-LD specifications.

The event aimed to validate a variety of scenarios, such as entity creation, retrieval, and querying—especially with a focus on distributed operations between brokers. The expected outcomes included identifying implementation bugs, fixing compatibility issues, and clarifying ambiguous points in the technical specifications. Another goal was to spot gaps in the current test suite, enabling its extension or correction for future test sessions. The analysis of interactions between different systems provided valuable insights for establishing common practices and improving the overall robustness of NGSI-LD-compliant tools.

Among the anticipated improvements for the NGSI-LD standard that emerged from the event were clearer semantics for attribute merging operations, a review of datasetId handling in distributed queries, and a more coherent structure for CSourceRegistration. Discussions also included the potential integration of new query language features, such as a "difilter" operator to filter datasets within entities. The Plugtest experience also highlighted the need to revise certain specification clauses (e.g., sections 5.6.17 and 5.6.18) to improve consistency and implementability—contributing to a clearer and more practical future evolution of the NGSI-LD standard for all involved stakeholders.

2 References

The following base specifications were partly validated or consulted in the Plugtest event:

[i.1] ETSI GS CIM 009 (V1.8.1) (2024-03): "Context Information Management (CIM); NGSI-LD API".

[i.2] ETSI GS CIM 054 (V.0.0.3) (2025-02): "Context Information Management (CIM); NGSI-LD Interoperability tests specification 3rd TTF"

3 Abbreviations

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

API	Application Programming Interface
CIM	Context Information Management
GS	Group Specification
IOP	InterOPerability
JSON	JavaScript Object Notation
JSON-LD	JSON Linked Data
NGSI	Next Generation Service Interface

NGSILD	Next Generation Service Interface Linked Data (same as NGSI-LD)
SUT	System Under Test

4 Participants

A total of 23 individuals participated in the NGSI-LD Plugtests event. This included 17 onsite participants, 1 remote participant, 1 technical expert, and 4 ETSI staff members. All participants were involved in the execution of the test cases and related technical sessions.

Table 4-1 shows the list of the organizations which participated during the event along with their role.

Table 4-1: Participants

Company Name	Role
CNIT – Consorzio Nazionale Interuniversitario per le Telecomunicazioni	Technical Expert
EGM	Implementer
Fraunhofer IIS/EAS	Application
FIWARE Foundation e.V.	Implementer
ETSI	Host
IUDX, IISc	Test tool provider
KEREVAL	Test tool provider
KETI – Korea Electronics Technology Institute	Implementer
NEC Laboratories Europe GmbH	Implementer
Sfera s.r.l.	TTF Experts

5 Details about the event

5.1 Scope and objectives

NGSI-LD (specified in ETSI GS CIM 009 [i.1]) is a standard for sharing and exchanging context information in a distributed environment. The key aspects of the specification include Information Model, API, JSON-LD serialization, and security/privacy. The goal of NGSI-LD is to enable interoperability among diverse context information systems and applications, supporting use cases in smart cities, IoT, and other domains.

Interoperability events play a crucial role as a cost-efficient and effective solution to achieve the goals of ensuring seamless data exchange between applications and brokers of context information. The testing will cover Interoperability Tests (GS CIM-054), Conformance Tests (GS CIM-029 and GS CIM-013), and Distributed operations (GS CIM-053).

A lot of emphasis would be put to covering the GS CIM 054. This is a necessity since, during the past few years, several NGSI-LD Context Brokers and related software modules, such as data ingestion, event processing, and visualization, are being developed globally, reflecting the widespread adoption of NGSI-LD standards. With multiple open-source implementations available, establishing comprehensive interoperability test cases is essential to ensure that different brokers, even from various vendors, can operate together seamlessly. While these tests do not directly validate the NGSI-LD specification, these tests may also be used to demonstrate whether the implementations can interoperate effectively and whether the NGSI-LD specification should be revised to avoid ambiguity (if any). The results can be applied in specific scenarios to highlight system compatibility, showcase cross-vendor collaboration, and provide valuable feedback for improving overall implementation quality.

To overcome these challenges, there is a pressing need for a structured and systematic approach to interoperability validation. This involves developing a comprehensive set of test cases and scenarios that simulate realistic operational contexts and stress the interactions between different brokers and NGSI-LD components. These tests should go beyond basic conformance to the specification and focus on verifying that systems can exchange and process data correctly in distributed and cross-vendor environments. Additionally, the testing framework must be flexible enough to highlight discrepancies, detect edge cases, and support updates as the NGSI-LD specification evolves.

All above points were the main focus during the NGSI-LD Plugtests hosted by ETSI, where multiple implementations are tested against a shared suite of scenarios and validation tools. This event served as neutral ground for identifying technical mismatches, refining implementation practices, and fostering dialogue between developers and standardization bodies. The tests were not intended to certify compliance but rather to explore how well systems

interoperate and to identify areas in the NGSI-LD specification that may require clarification or revision. Ultimately, these efforts aimed to contribute to building a more robust, compatible, and future-proof NGSI-LD ecosystem.

5.2 Network Architecture

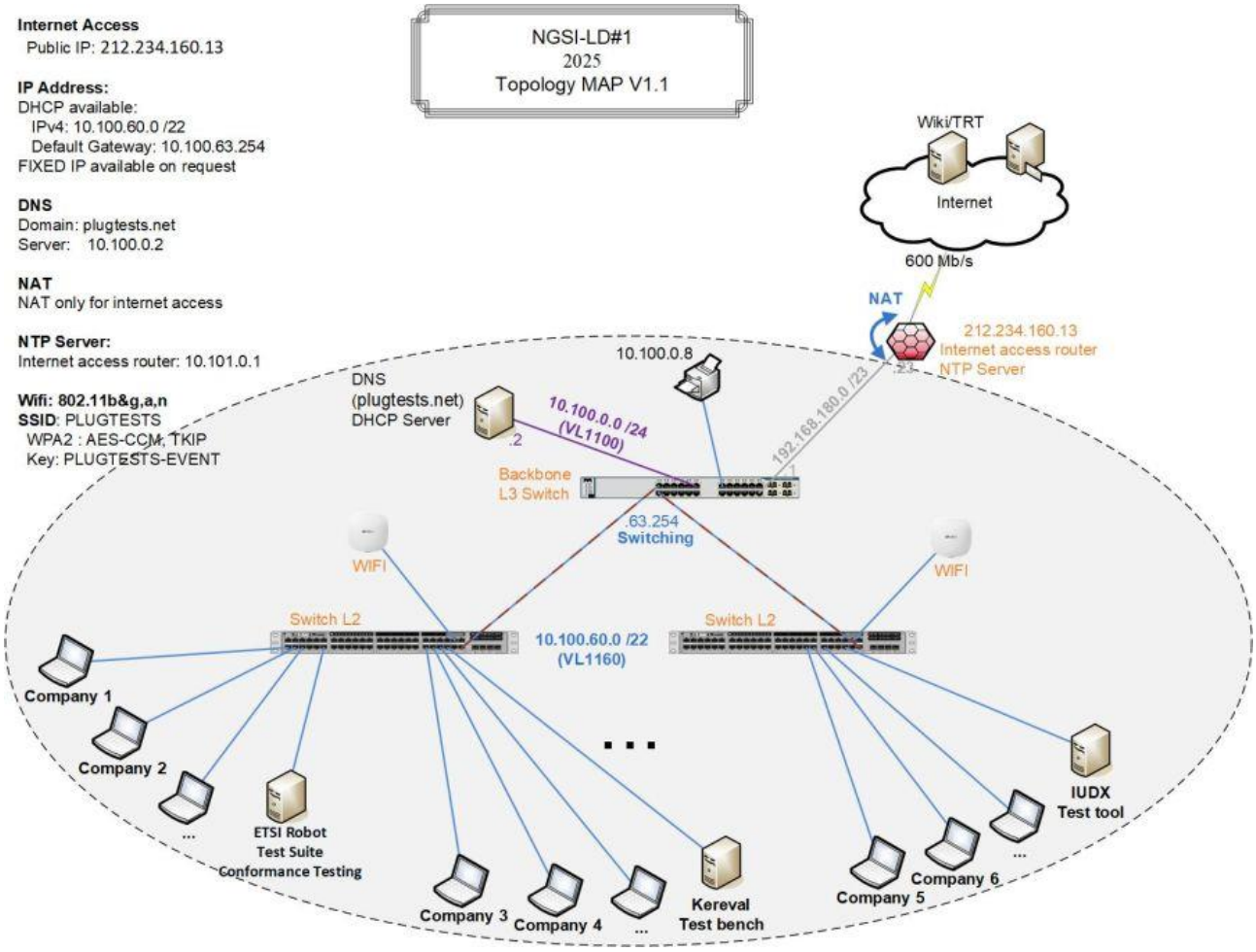

Figure 5.2-1: Network Architecture

Figure 5.2-1 shows the network architecture set up for the NGSI-LD Plugtests event.

To simplify the communication between the services run by the participants, IPs from 10.100.60.1 to 10.100.60.38 were provided as static IPs to the participants. For each participant, a DNS was created using the first name of the participant itself.

5.3 Conformance Tests

5.3.1 Configurations

Test configurations are defined upon the different architectures' options defined in clause 4.3 of ETSI GS CIM 009 [i.1]. Considered architectures are

- **Centralized architecture:** A Central Broker stores all the context information. There are Context Producers that use update operations to update the context information in the Central Broker and there are Context Consumers that request context information from the Central Broker, either using synchronous one-time query or asynchronous subscribe/notify operations.

- **Distributed architecture:** All information is stored by the Context Sources. Context Sources implement the query and subscription part of the NGSI-LD API as a Context Broker does. They register themselves with the Context Registry, providing information about what context information they can provide, but not the context information itself.
- **Federated architecture:** The architecture works in the same way as the distributed architecture described in clause 4.3.3 of ETSI GS CIM 009 [i.1], except that instead of simple Context Sources, whole domains are registered with the respective Context Broker as point of access. Typically, the domains will be registered to the federation Context Registry on a more coarse-grained level, providing scopes, in particular geographic scopes, that can then be matched to the scopes provided in the requests.

Test configurations are defined to test different entities such as NGSI-LD Broker, NGSI-LD Context Producer, NGSI-LD Context Consumer, NGSI-LD Context Source, etc.

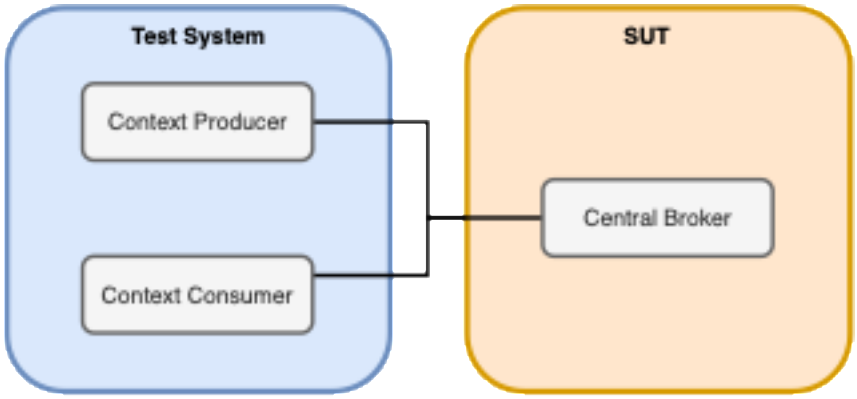

Figure 5.3.1-1: Test configuration 1 (CF_01)

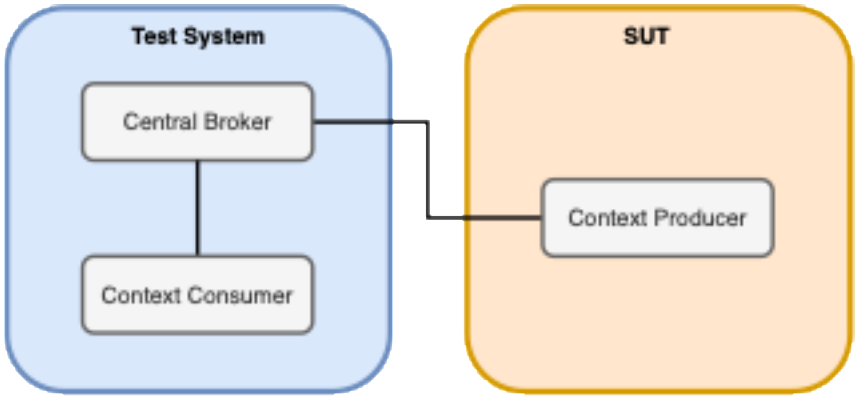

Figure 5.3.1-2: Test configuration 2 (CF_02)

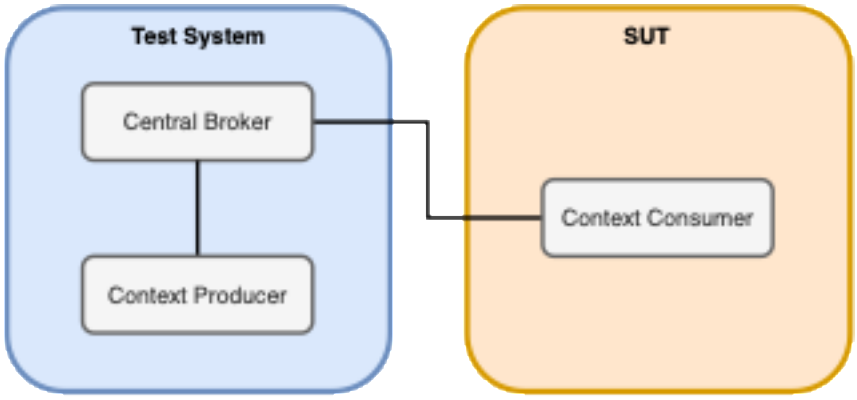

Figure 5.3.1-3: Test configuration 3 (CF_03)

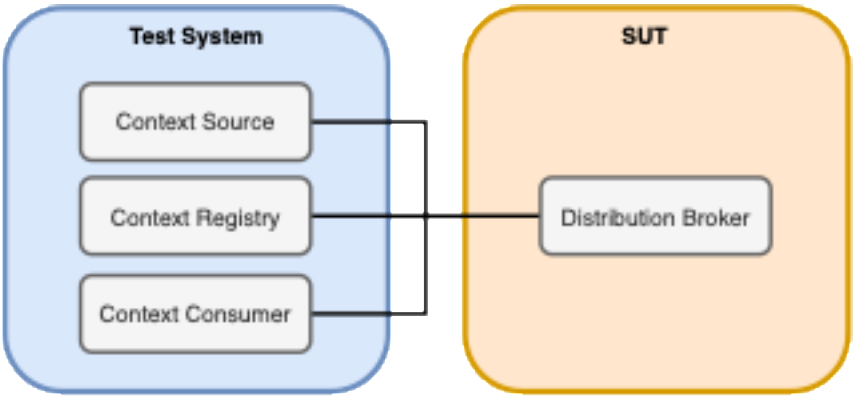

Figure 5.4.1-4: Test configuration 4 (CF_04)

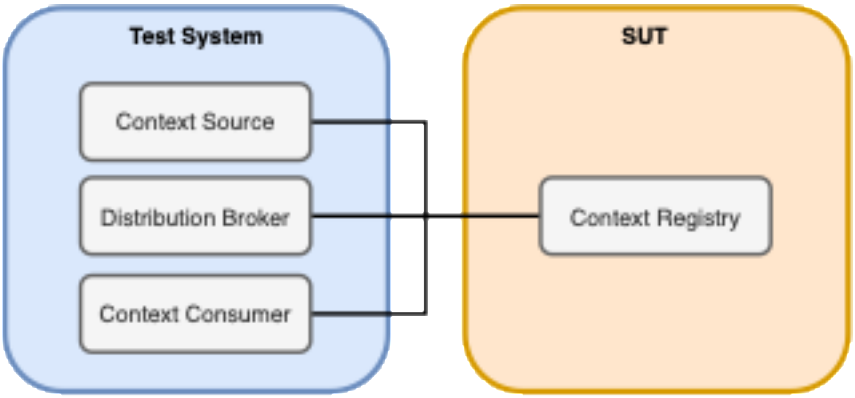

Figure 5.5.1-5: Test configuration 5 (CF_05)

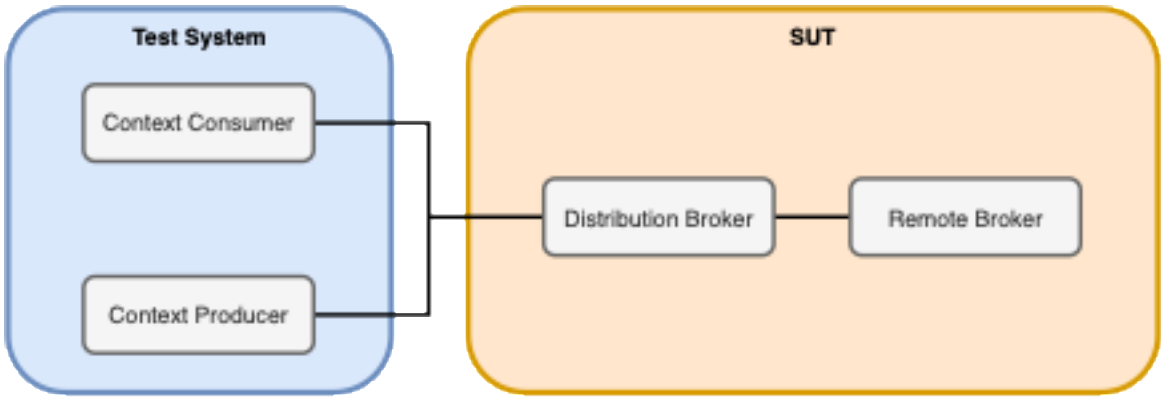

Figure 5.6.1-6: Test configuration 6 (CF_06)

5.3.2 Test tools

5.3.2.1 ETSI's Robot Framework

The ETSI Robot Framework is an automated testing framework specifically designed to support conformance testing of NGSI-LD implementations against the ETSI GS CIM 009 [i.1] specification. It is capable of executing a wide range of test cases to verify whether an NGSI-LD Context Broker behaves according to the standard's requirements (currently up to version v1.6, but the framework is being constantly updated). The framework covers both local operations—where a single broker handles all entity management tasks—and distributed scenarios involving interactions between multiple brokers. By simulating various API calls and validating responses against expected outputs, the Robot Framework plays a key role in ensuring the correctness, consistency, and reliability of NGSI-LD implementations.

With respect to clause 5.3.1 of the present document, this test suite covers CF_01, CF_02, CF_03, CF_04 and CF_06.

5.3.2.2 KEREVAL ITB Platform

The Kereval ITB Platform is a conformance testing platform developed to assess NGSI-LD implementations against the ETSI GS CIM 009 [i.1] specification. It focuses on validating the correct behavior of a single NGSI-LD Context Broker in local operation scenarios. The suite executes a structured set of test cases that check compliance with the standard's functional requirements, including entity creation, updates, deletions, and querying. Unlike the ETSI Robot Framework, the Kereval suite does not support distributed use cases involving multiple brokers, and is therefore limited to evaluating standalone broker functionality within isolated deployments.

With respect to clause 5.3.1 of the present document, this test suite covers CF_01, CF_02 and CF_03.

5.3.2.3 IUDX's Postman collection

The IUDX Postman Collection is a lightweight conformance testing tool designed to validate the behavior of NGSI-LD Context Brokers through a series of predefined API requests. It focuses on testing local broker operations by executing various HTTP calls and using embedded Postman scripts to verify the correctness of responses. While it enables functional checks for standard operations like entity creation, retrieval, and querying, its capabilities are limited to local (non-distributed) scenarios. The use of Postman scripting allows for basic validation logic, but the tool lacks the advanced automation and coverage provided by more comprehensive test frameworks.

With respect to clause 5.3.1 of the present document, this test suite covers CF_01, CF_02 and CF_03.

5.4 Interoperability Tests

5.4.1 Test Data

The test data was taken from clause 4.1 of [i.2].

5.4.2 Configurations

5.4.2.0 Foreword

The following clauses provide a detailed description of the specific test configurations that has been employed during the interoperability tests in the ETSI Plugtests event. Each clause includes a diagram illustrating the topology of the configuration, a formal description that explains the rationale for selecting the given configuration.

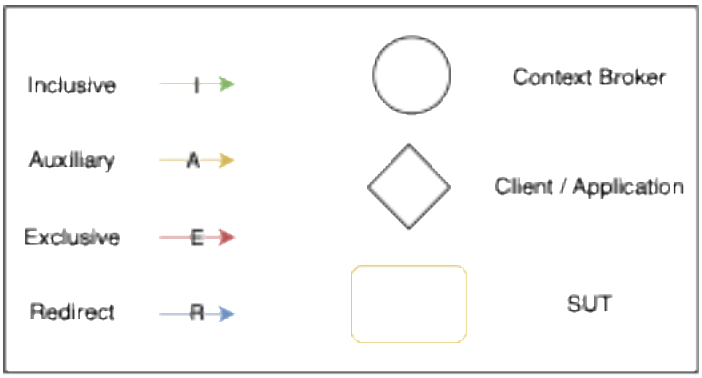

Figure 5.4.2.0-1: Legend

Figure 5.4.2.0-1 illustrates the legend for each element utilized in the diagrams of the configurations described in the present document. As depicted, registrations are represented as arcs, characterized by a color and a label that denote a specific Context Source Registration mode. Nodes are depicted as circles. Additionally, a dashed line is employed to indicate interactions with a specific node originating from an element external to the System Under Test (SUT), typically a client.

For more details about the technical aspects related to each configuration one can consult clause 4.2 of [i.2].

5.4.2.1 IOP_CNF_01: Three brokers with Inclusive and Exclusive

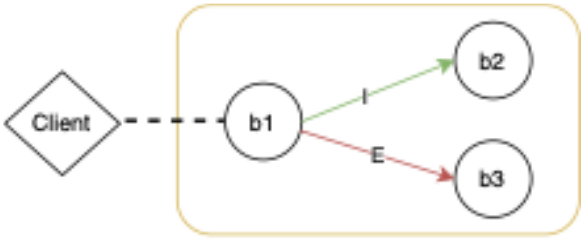

Figure 5.4.2.1-1: IOP_CNF_01 topology with registrations

Figure 5.4.2.1-1 shows the topology for IOP_CNF_01 which is one of the simplest possible setups, consisting of only three brokers. It is suitable for both Provision and Consumption operations, testing a scenario where broker b2 mirrors most of the information contained in b1 and b3 contains some exclusive data that cannot be stored in b1.

5.4.2.2 IOP_CNF_02: Four brokers with Inclusive and Redirect

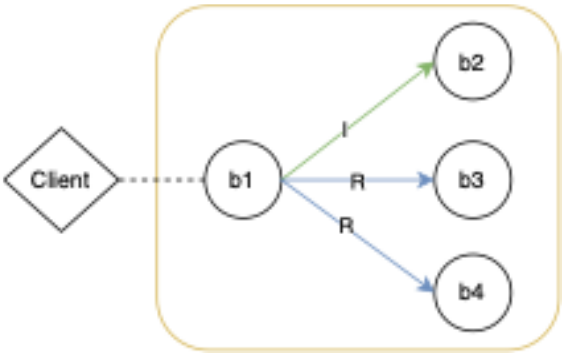

Figure 5.4.2.2-1: IOP_CNF_02 topology with registrations

Figure 5.4.2.2-1 shows the topology for IOP_CNF_02 which is a variant of IOP_CNF_01 with four brokers and Redirect registrations. It leverages the same approach, but it allows for more flexible tests. This is justified by the fact that Redirect registrations don't require to specify an Entity id, but can work with Entity types.

5.4.2.3 IOP_CNF_03: Four brokers with Auxiliary and Inclusive

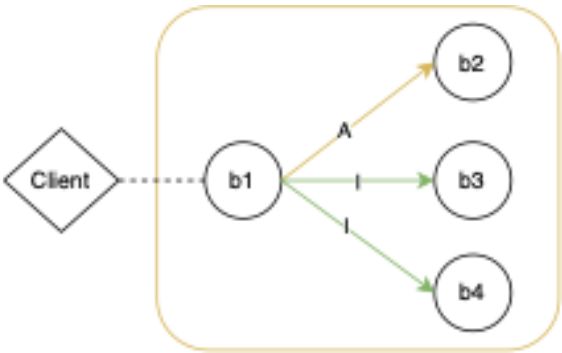

Figure 5.4.2.3-1: IOP_CNF_03 topology with registrations

Figure 5.4.2.3-1 shows the topology for IOP_CNF_03 which is a variant of IOP_CNF_02 with Auxiliary and Inclusive registrations. Considering the presence of an Auxiliary registration, this configuration is very suitable for testing

Consumption operations. Provision operations can be tested as well, but the focus shifts towards a more “negative” approach where the objective is checking that an operation was **not** performed when specific requirements are met.

5.4.2.4 IOP_CNF_04: Five brokers with all registration modes

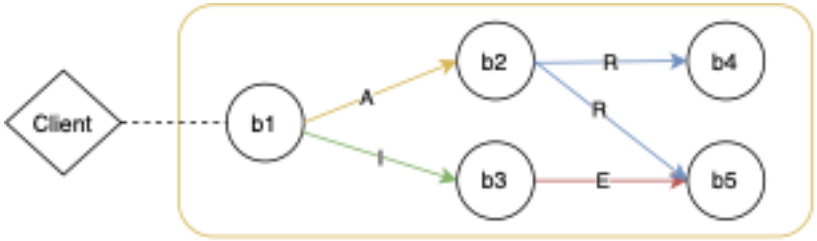

Figure 5.4.2.4-1: IOP_CNF_04 topology with registrations

Figure 5.4.2.4-1 shows the topology for IOP_CNF_04. This configuration contains all registration modes, enabling the execution of complex and comprehensive tests, particularly for evaluating Consumption operations. However, it is less suitable for Provision operations due to the presence of the Auxiliary registration from b1 to b2. In the case of a Provision request, b1 will not contact b2, which implies that nodes b2, b4, and potentially b5 (depending on the specific request) will remain unaffected.

5.4.2.5 Operations and Test cases

The following operations of [i.1] were covered:

- Create Entity
- Retrieve Entity
- Query Entities

For “Create Entity” the following test cases were run:

- Create OffStreetParking:1
- Create OffstreetParking:2
- Create Partial OffstreetParking:1

For “Retrieve Entity” the following test cases were run:

- Retrieve OffstreetParking:1
- Retrieve OffstreetParking:2
- Retrieve “location” from OffstreetParking:1

For “Query Entities” the following test cases were run:

- Query Entities with type OffstreetParking (GET)

- Query Entities with type OffstreetParking (POST)
- Query the “location” Property from Entities having type OffstreetParking or Vehicle (GET)

6 Achieved Results

6.1 Observations

6.1.1 General

The test campaign was structured around four major interoperability configurations (IOP_CNF_01 to IOP_CNF_04), each designed to explore different aspects of NGSI-LD operations in a controlled, replicable way. These configurations aimed to assess not just standard CRUD operations but also the brokers’ ability to manage partial updates and respond accurately to both simple and advanced queries—including type-specific and attribute-filtered requests.

The brokers selected represent a mix of widely adopted open-source NGSI-LD implementations and academic or institutional prototypes. This diversity helped surface variations in interpretation of the specification and implementation maturity, both of which are crucial for understanding real-world interoperability challenges.

Approximately 20 permutations of configurations and test cases were applied, leading to a total execution of roughly 120 individual test scenarios. These covered a week of continuous testing and debugging efforts, which also allowed teams to fix implementation bugs on the fly and clarify ambiguities in the specification.

The tools used for this purpose—ETSI’s Robot Framework, Kereval’s ITB Platform, and the IUDX Postman Collection—each served a unique role. The Robot Framework, as the most comprehensive tool, supported both local and distributed testing in alignment with ETSI GS CIM 009 [i.1]. Kereval’s suite provided detailed local conformance testing, while the IUDX tests, although limited to local scenarios, were valuable for validating API responses through Postman scripts.

Special attention was given to brokers capable of distributed operations, namely Orion-LD and Scorpio. These were prioritized for tests involving federation and multi-broker interactions, a critical feature of NGSI-LD’s scalability model. Stellio was considered as a third candidate for distributed scenarios depending on time constraints. Conversely, brokers like Fiware Lepus and KETI, which do not support distributed mode, were excluded from configurations requiring federation functionality (but were always involved as the “leaf” broker of a federated test case).

Due to the specific nature of distributed interoperability, only brokers fully implementing the necessary features could participate in those scenarios. This constraint naturally reduced the number of valid broker combinations that could be explored in distributed mode, limiting the overall permutation space compared to local-only testing.

6.1.2 Required actions taken to run the Interoperability tests

Based on the nature of the tests, only the implementations that have implemented distributed operations could be used as distribution brokers. This narrowed down the eligible brokers only to Orion-LD, Scorpio and Stellio instances. This also reduced the total number of permutations that it was possible to effectively run.

It was required to craft the following user @context to run the tests against City Data Hub.

```
{
   "@context": {
      "Bus": "https://ngsi-ld-test-suite/context#Bus",
      "isParked": "https://ngsi-ld-test-suite/context#isParked",
      "isNextToBuilding": "https://ngsi-ld-test-suite/context#isNextToBuilding",
      "source": "https://ngsi-ld-test-suite/context#source",
```

```
    "providedBy": "https://ngsi-ld-test-suite/context#providedBy",
    "availableSpotNumber": "https://ngsi-ld-test-suite/context#availableSpotNumber",
    "totalSpotsNumber": "https://ngsi-ld-test-suite/context#totalSpotNumber",
    "availableSpotsNumber": "https://ngsi-ld-test-suite/context#availableSpotsNumber",
    "totalSpotNumber": "https://ngsi-ld-test-suite/context#totalSpotsNumber",
    "isNextToBuilding": "https://ngsi-ld-test-suite/context#isNextToBuilding",
    "reliability": "https://ngsi-ld-test-suite/context#reliability",
    "hasWeatherObserved": "https://ngsi-ld-test-suite/context#hasWeatherObserved"
  }
}
```


6.1.3 Bugs fixed in implementations

The following bugs were identified and successfully fixed during the event:

City Data Hub

- The @vocab was not applied correctly as per core context
- The “ngsi-ld/v1” path was missing

Orion-LD

- Distributed POST query was not implemented
- Fixed the timeout in Orion-LD being shorter than expected
- Added support to multitype queries with OR operators (pipe character)
- Solved an issue with OrionLD not downloading the @context hosted by KETI. (user agent missing)

Scorpio

- There was an issue on Scorpio when retrieving an Entity by ID while assuming entityMaps are supported, even when the remote broker does not support them
- Fixed an issue regarding the *types* compaction mechanism not being applied when running distributed operations

Stellio

- Distributed POST query was not implemented
- Fixed *attrs* names not being forwarded in distributed operation

6.1.4 Points that were clarified during the event

The ITB platform was initially unaware of the logic implemented by Scorpio concerning the temporal representation of entities inserted through the standard Core API, resulting in test failures. To address this, a startup flag can now be specified to disable this feature during testing.

Several discussions have been held between members of the ISG CIM group and the IUDX team to update and adapt the IUDX Conformance tests (Postman Collection). These adaptations ensure compliance with NGSI-LD standards and compatibility when executed against an NGSI-LD broker.

6.1.5 Points that may require changes on ETSI GS CIM 009

6.1.5.1 Clarifications on 5.6.17 and 5.6.18

In clauses Replace Entity (5.6.18) and Merge Entity (5.6.17) of [i.1], the Behaviour section should be revised to separately address "exclusive" and "redirect" cases. First, handle the "exclusive" logic, then remove any relevant data, and finally address the "redirect" logic—following the same structure used in the Create Entity clause. Additionally, ensure it is clearly stated that an error must be returned if the user attempts to create a Redirect for a local resource.

6.1.5.2 Query Entities by IDs is not allowed

Querying entities using only IDs without specifying a type is currently not allowed. It was agreed that further discussion is needed on this point. The general consensus was to consider removing the requirement of including the type when IDs are provided.

6.1.5.3 Replace propertyNames and relationshipNames with attributesNames

In CSourceRegistration, the fields "propertyNames" and "relationshipNames" are used to target specific attributes. It was noted that there is always a risk of name collisions between Properties and Relationships in distributed scenarios. Furthermore, this distinction is typically lost when brokers interact with distributed brokers. The general consensus was to replace both fields with a unified field: "attrsNames".

6.1.5.4 Clarify what to do while merging two Attributes when one of them doesn’t have observedAt

When merging two instances of the same Attribute (as a result of a distributed request), the current specifications are unclear on how to proceed if only one instance includes the observedAt TemporalProperty. Two possible solutions have been proposed:

- Retain the instance with observedAt
- Retain the most recent instance, based on their modifiedAt timestamp

Further discussion is required to reach a final decision on this matter.

6.1.5.5 Merging Reported Errors

When multiple 404 errors are returned, the federation broker merges these responses to simplify the experience for the end user. For example, if an entity deletion request is sent and two brokers return 200 OK but a third returns a 404 NOT FOUND, the federation broker suppresses the 404. This approach avoids overwhelming the user with redundant errors, especially in scenarios involving many brokers.

6.1.5.6 Matching algorithm

The matching algorithm for registrations is currently complex and insufficiently defined in the specification. For instance, how should the system behave if a user provides an idPattern and the registrations also use idPatterns? Further clarification and guidance are needed here.

6.1.5.7 Except/Omit

Consider the concept of "except/omit" registrations—where a registration includes all attributes except for a specific Property. This use case should be evaluated for possible inclusion and specification.

6.1.5.8 Clarify Auxiliary Registrations Behaviour

The behaviour of auxiliary registrations needs to be explicitly defined. Clear guidelines should be provided on how they interact with other registrations, and whether they are allowed when proxied registrations are used.

6.1.5.9 Add parameter to RegistrationManagementInfo

The CSourceRegistration object should include a new "limit" field. This field indicates the maximum extent to which the source should be used, helping to manage load distribution and query planning more effectively.

6.1.6 Interoperability with the IUDX application

6.1.6.1 Foreword

It was determined that the data available in IUDX could be aligned with NGSI-LD concepts through the following conceptual adaptations. Three possible approaches were identified, as detailed below. In all cases, the dataset "TransitManagement" was categorized as a type. Its data structure corresponded to the TransitManagement Data (NGSI-LD Format) described in section 2.2.

6.1.6.2 Type-1

The TransitManagement dataset is treated as a single entity with multiple "datasets" with "datasetId" <buses> as there are different buses publishing data. Each bus has a unique identifier called license-plate which is used as the datasetId for better temporal evolution.

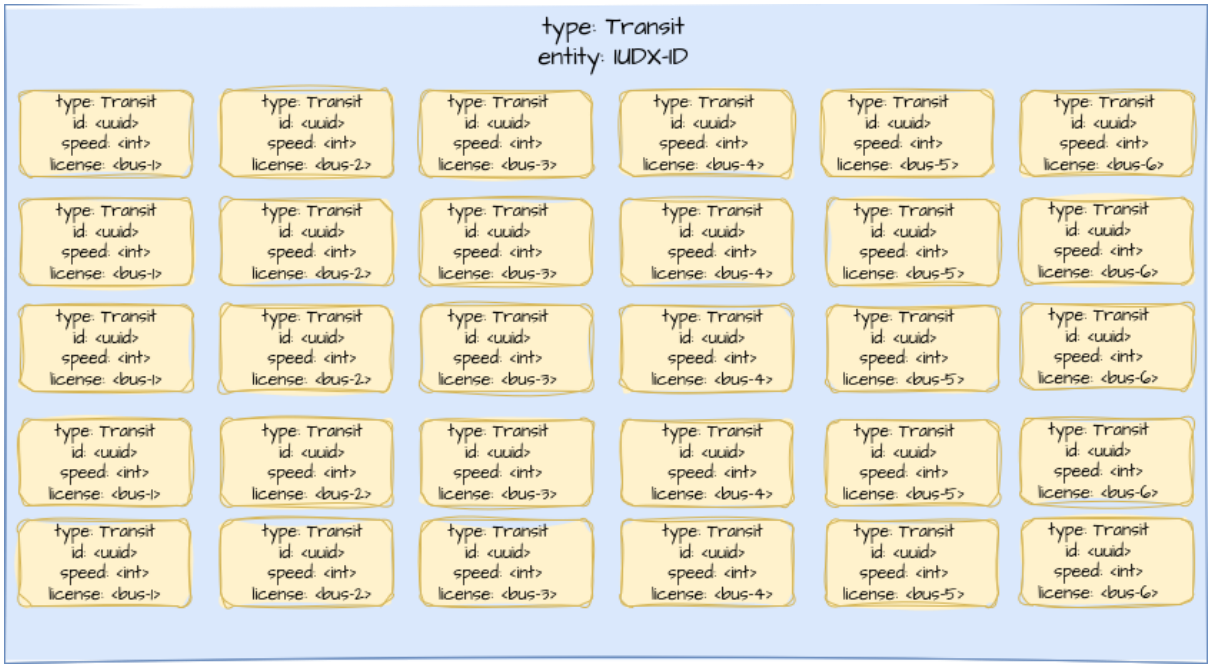

Figure 6.1.6.2-1: Type-1 schema

Observations

- In this case, we have each observation with a dataset-id added/upserted to an entity
- All the observation will have the same same type "transitManagement" and "entity-id"
- The query happens at the "entity-id" level where types are hidden.
- No change required in IUDX flow. Each resource which has an IUDX "id" will be an NGSI-LD "entity"
- The query will happen with "entity"="<entity-id>" as it is now
- Authorization can happen at an entity level
- Adding datasetId to all the observed values increases the data size
- The temporal + spatial + attribute query runs only on the latest data. Due to this there are limitations on the queryability aspects. For example, if the timestamp is
- But this approach discovered a difficulty in the NGSI-LD API in querying and filtering by datasetId, which in turn suggested some possible improvements to the API.

6.1.6.3 Type-2

The TransitManagement dataset is treated as a single type with multiple entities where each “observation” will be an “entity”. The resources <buses> shall be different buses publishing data. Each entity <an observation of a bus> has a unique identifier called license-plate, observedAt etc. type associated with it for temporal evolution.

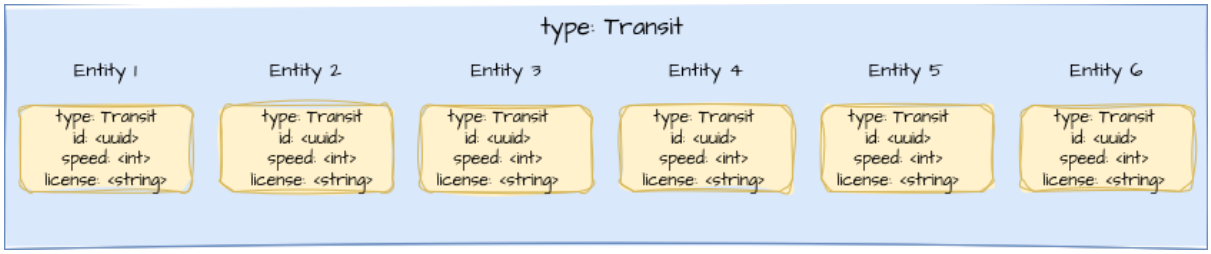

Figure 6.1.6.3-1: Type-2 schema

In this case, the temporal + spatial queries will be as follows:

```
curl --location --globoff 'http://broker/ngsi-
ld/v1/temporal/entities?type=TransitManagement&geoproperty=location&georel=within&geometry=Polygon
&coordinates=[[[72.76%2C21.15]%2C[72.76%2C21.13]%2C[72.78%2C21.13]%2C[72.76%2C21.15]]]&tim
erel=between&timeAt=2020-09-25T07%3A36%3A35Z&endTimeAt=2020-09-
25T08%3A00%3A35Z&q=speed<30'
```

Figure 6.1.6.3-2: Temporal curl request for Type-2

The response will contain all the entities within the time range where the latest data of a given entity is validated for geo queries and filtered by attribute. If it falls within the range, the entity is added into the response.

Observations

- In this case, many entities are created with the same type “transitManagement”
- The query could happen at the type level where entities are hidden.
- Each resource which has an IUDX “id” should be moved to a NGSI-LD “type” represented as “type”: “<id>”
- The query would use “type”=“<id>” instead of “id”=”<resourceId>”
- Authorization could happen at the type level

6.1.6.4 Type-3

In this case, the dataset “TransitManagement” is categorised as type. Additionally, it would be desiderable to introduce the possibility of using “license-plate” as an entity with “license-plate” used as the “entity-id”. The data structure of it is as shown in the TransitManagement Data (NGSI-LD Format) in section 2.2.

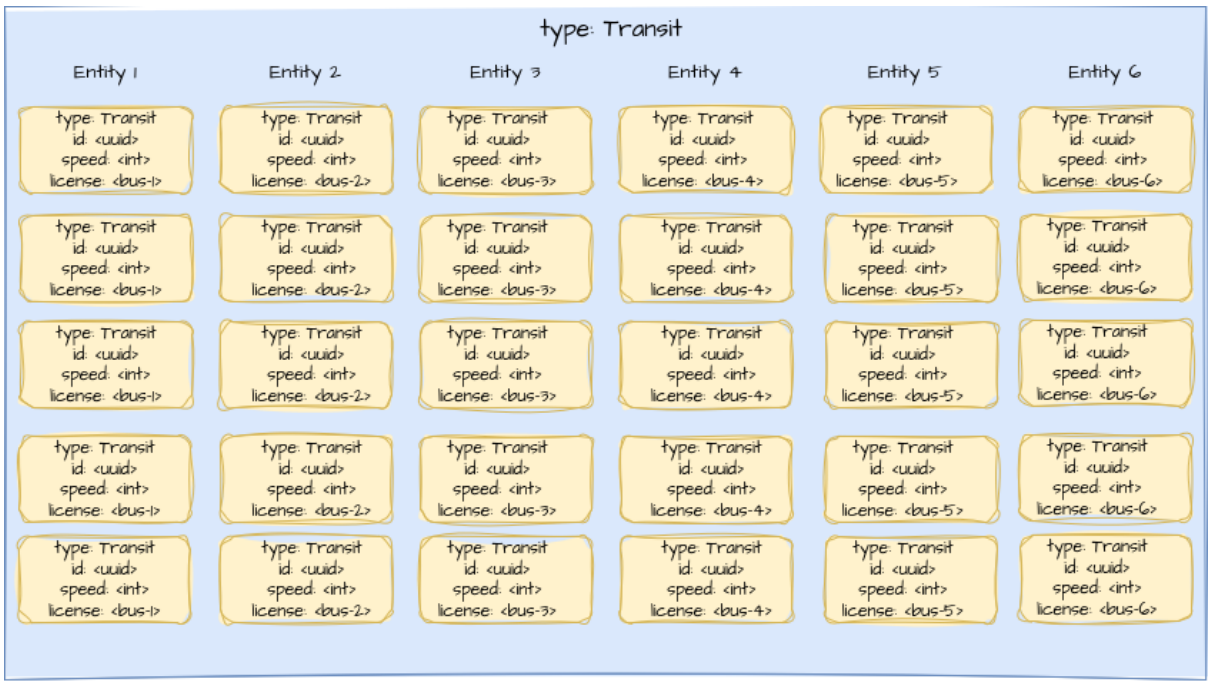

Figure 6.1.6.4-1: Type-3 schema

In this case, the temporal + spatial queries will be as follows:

```
curl --location --globoff 'http://martin:9090/ngsi-ld/v1/temporal/entities?type=TransitManagement&geoproperty=location&georel=within&geometry=Polygon&coordinates=[[[72.76%2C21.15]%2C[72.76%2C21.13]%2C[72.78%2C21.13]%2C[72.76%2C21.15]]]&timerel=between&timeAt=2020-09-25T07%3A36%3A35Z&endTimeAt=2020-09-25T08%3A00%3A35Z&q=speed<30'
```

Figure 6.1.6.4-2: Temporal curl request for Type-3

The response will contain all the entities within the time range where the latest data of a given entity is validated for geo queries and filtered by attribute. If it falls within the range, the entity is added into the response.

What this model will also allow is operations on each entity. This will allow an additional level of granularity/filterering if the user wishes to look into. For example, a temporal/spatial/attribute query will be possible for each bus as a bus is modeled as an entity here.

Observations

- In this case, we have limited entities created <one per license_plate> with the same type “transitManagement”
- The query happens at the type level where entities are hidden. In this case, the response will contain all the bus data.
    - Each resource which has an IUDX “id” should be moved to a NGSI-LD “type” represented as “type”: “<id>”
    - The query will happen with “type”=“<id>” instead of “id”=”<resourceId>”
    - Authorization can happen at a type level
- The query can also happen at the entity level where the response will contain only the given entity
    - Each bus can be represented as an entity with “license_plate” as the “entity-id”
    - The query can happen with “id”=”<license-plate>”
    - Authorization can happen at a type level

6.1.6.5 Impact on API: new functionality on datasetId filtering

IUDX utilized the NGSI-LD Entity as an entry point to a “data provider,” rather than representing each individual data item as a separate Entity. This approach allowed them to retain existing data elements as opaque structures within a larger array, preserving their internal format instead of fully converting them into NGSI-LD Entities.

As a result, their access control mechanism was Entity-based. For example, a single Bus Entity effectively acted as a wrapper for all buses. Performing a GET request on the Bus Entity would return a complex JSON object containing data for all buses.

Initially, the proposed approach was to assign a datasetId to each bus individually. However, this led to issues during filtering. Specifically, a q query in NGSI-LD functions as an Entity selector rather than a selector for specific datasetId values within an Entity. Therefore, filtering based on datasetId values did not function as intended.

For temporal filtering, datasetIds outside the requested time interval are removed before the q query is applied. However, to fully support the IUDX use case—i.e., filtering each Entity based on conditions applied to its individual datasetId entries—a new mechanism is required.

This proposed mechanism, referred to as difilter, would support the full NGSI-LD query language. It would operate either on a specific Entity (in the Retrieve Entity operation) or as a secondary filtering step in the Query Entities operation, following the application of the q filter.

The revised filtering process would follow these steps:

1) Temporally filter Entities based on the requested interval.
2) Select Entities matching the q filter.
3) Apply pick and omit operations.
4) Apply the difilter expression within each Entity to extract only the datasetId entries that match the specified conditions.

The difilter expression may differ from the q filter or mirror it, depending on a flag configuration. Additionally, the difilter should support a slicing flag, indicating that the selected datasetIds must share the same value. This enables the selection of attributes measured by the same device (i.e., having the same datasetId).

An alternative approach would involve creating one separate NGSI-LD Entity per bus, rather than mapping each distinct bus to a datasetId within a single Entity. This method is more aligned with NGSI-LD best practices.

Architecturally, IUDX could maintain its current deployment while introducing an additional, fully-fledged Context Broker. This broker would handle individual data items, injecting each as a separate NGSI-LD Entity.

Each Entity would be assigned a unique identifier—such as the vehicle’s license plate—and a type, for example, "Bus", to elevate it to a properly defined NGSI-LD Entity.

To preserve existing access control mechanisms, IUDX could implement a wrapper around the Context Broker. This wrapper would manage authorization and translate Retrieve Entity operations into Query Entities requests, which would then be forwarded to the underlying Context Broker.

6.1.6.6 OR behavior of the datasetId matching

The matching of datasetId is an OR: as soon as any Entity's datasetId's value matches the "q" query, the Entity is selected as matching.

This means that if i want Cars with speed > 25, then if a Car has two speedometers that measured 23 and 27, BOTH values are presented in the resulting Entity.

We could think about having an option for also doing it as an AND. This would mean that the Entity is returned if all the "measuring devices" are in agreement about the measured value.

This said, we acknowledge that this does not solve IUDX's problem because it will return ALL buses if they match the speed (or the geographic area) or NONE, if even one of them is outside the range.

6.1.7 Issues found in the Robot Framework Conformance Test Suite

- Trailing zeros in numerical values in the results of aggregated temporal queries fail the tests (not handled by the TemporalPropertyOperator implemented in the Test Suite)
- A Test Case should not fail if a subscription has the default value for the notificationTrigger and the Context Broker does not include it in the responses
- The expectations on the Query object in the Test Case 037_04 are incorrect
- The Test Case 037_09 has to be checked for the way the datetimes (startAt, endAt) are checked in the actual response
- The replacement of the identifiers of the Context Source Registration in the expectation is incorrect in the Test Case 037_10 and thus fails the test.
- The checks made on the pagination of results in the Test Case 037_11 are not sufficient and should be improved
- The expectations in the Test Case 026_01 seem incorrect and have to be checked carefully
- Some Test Cases related to the JsonLd Context Server do a reload of the core context but this is not an allowed operation
- And more generally:
    - Expectations should be relaxed on optional members (notificationTrigger, timesSent, status, ...) that may not be provided by context brokers when they have the default value
    - Use simpler payloads in the Test Case to avoid failures when part of a data type is not implemented (for instance datasetId in Attributes, or timeInterval in Subscriptions)
- Mock server for distributed test cases would be better since the SuT should only be the Distribution Broker
    - An issue has to be addressed with the current Mock Server library adopted: running “Start Server” a second time overrides the first call, i.e. it is not possible to listen on two (or more) different ips/ports

6.2 Statistics

Regarding the Interoperability tests, the statistics shown in the following figures were returned by the TRT platform.

Number of Sessions: 335
Of the 335 reported sessions 12 were agreed (3.6 %)

Interoperability		Not Executed		Totals	
OK	NO	NA	OT	Run	Results
88 (87.1%)	13 (12.9%)	110 (52.1%)	(0.0%)	101 (47.9%)	211

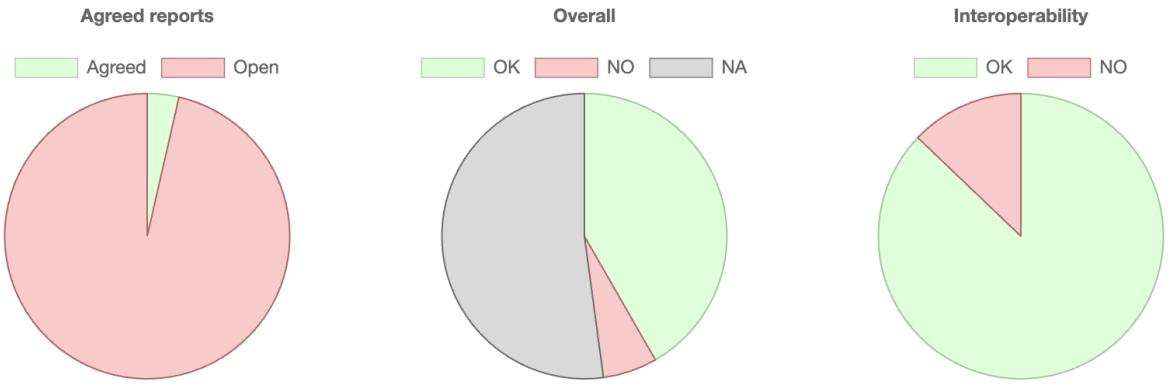

Figure 6.2-1: Overall Result

	Interoperability		Not Executed		Totals
	OK	NO	NA	OT	Run
Minimum	2	1	1	0	3
Maximum	6	3	6	0	6
Mean	4.7	2.0	3.5	0.0	5.5
Deviation	1.35	1.00	2.50	0.00	0.89

Figure 6.2-2: Statistics per Session

	Interoperability		Not Executed		Totals		
	OK	NO	NA	OT	Run	Results	Chart
IOP_CNF_03	15 (83.3%)	3 (16.7%)	0 (0.0%)	0 (0.0%)	18 (100.0%)	18	
IOP_CNF_02	18 (85.7%)	3 (14.3%)	79 (79.0%)	0 (0.0%)	21 (21.0%)	100	
IOP_CNF_01	36 (85.7%)	6 (14.3%)	30 (41.7%)	0 (0.0%)	42 (58.3%)	72	
IOP_CNF_04	19 (95.0%)	1 (5.0%)	1 (4.8%)	0 (0.0%)	20 (95.2%)	21	

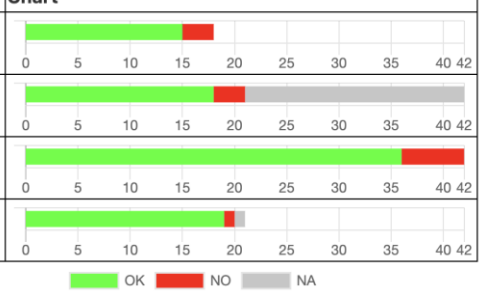

Figure 6.2-3: Results per group

	Interoperability		Not Executed		Totals	
	OK	NO	NA	OT	Run	Results
Create OffStreetParking:1	19 (100.0%)	0 (0.0%)	18 (48.6%)	0 (0.0%)	19 (51.4%)	37
Create OffStreetParking:2	15 (100.0%)	0 (0.0%)	17 (53.1%)	0 (0.0%)	15 (46.9%)	32
Retrieve OffStreetParking:1	16 (84.2%)	3 (15.8%)	17 (47.2%)	0 (0.0%)	19 (52.8%)	36
Retrieve OffStreetParking:2	6 (75.0%)	2 (25.0%)	4 (33.3%)	0 (0.0%)	8 (66.7%)	12
Query Entities with type OffstreetParking (GET)	14 (82.4%)	3 (17.6%)	17 (50.0%)	0 (0.0%)	17 (50.0%)	34
Retrieve location from OffStreetParking:1	9 (90.0%)	1 (10.0%)	13 (56.5%)	0 (0.0%)	10 (43.5%)	23
Query Entities with type OffstreetParking (POST)	3 (75.0%)	1 (25.0%)	11 (73.3%)	0 (0.0%)	4 (26.7%)	15
Query the “location” Property from Entities having type OffstreetParking or Vehicle (GET)	3 (50.0%)	3 (50.0%)	13 (68.4%)	0 (0.0%)	6 (31.6%)	19
Create Partial OffStreetParking:1	3 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	3 (100.0%)	3

Figure 6.2-4: Results per test

Annex A: Interoperability Implementation Guide

A.1 Introduction

This guide has been developed as a non-normative annex to support implementers of NGSI-LD specifications by capturing lessons learned, clarifications, and best practices identified during the interoperability testing event. The goal is to improve the consistency and correctness of implementations across different platforms.

A.2 Scope

This guide addresses the practical aspects of implementing the NGSI-LD standard, based on findings and experiences gathered during the ETSI Plugtests interoperability event.

A.3 Common Issues Observed

A.3.0

During the interoperability testing event, several recurring issues were identified across different implementations. These issues highlight areas where the NGSI-LD specification may be misinterpreted or where further implementation guidance is needed.

A.3.1 Context Expansion and JSON-LD Processing

- **Problem:** Many participants experienced issues related to improper handling of the `@context` element, leading to failures in JSON-LD expansion and inconsistencies in payload interpretation.
- **Typical causes:**
  - Use of an incorrect or incomplete custom context.
  - Failure to resolve remote contexts (e.g., lack of internet access or CORS restrictions).
  - Context definitions missing required term definitions for properties or relationships.
- **Impact:** Data consumers were unable to parse or interpret incoming payloads correctly, resulting in failed entity creation, updates, or queries.

A.3.2 Misuse or Incomplete Definition of Relationships

- **Problem:** Relationships were sometimes used without proper object URIs or without explicitly defining the `type` field.

- **Typical causes:**
  - Developers confusing `Property` and `Relationship` constructs.
  - Lack of validation checks on the target entity of a relationship.
- **Impact:** Linking between entities failed, which compromised linked data navigation and query results (e.g., `temporal queries` or `query by relationship`).

A.3.3 Invalid or Unsupported GeoProperties

- **Problem:** Several payloads failed validation due to incorrect GeoJSON structures or unsupported geometry types.
- **Typical causes:**
  - Using coordinates without specifying `type: Point`, or providing coordinates in an invalid order.
  - Sending GeoProperties without the proper `"type": "GeoProperty"` declaration.
  - Implementations supporting only limited geometry types (e.g., only `Point` and not `Polygon` or `LineString`).
- **Impact:** Location-based queries (e.g., within a bounding box or near a point) returned incomplete or incorrect results.

A.3.4 Inconsistent Support for API Features

- **Problem:** Some implementations partially supported NGSI-LD API features such as filtering, pagination, or temporal queries.
- **Typical causes:**
  - Implementers prioritizing core CRUD operations and neglecting advanced features.
  - Lack of clear examples or test cases for complex query parameters.
- **Examples:**
  - Filtering using `q` or `georel` parameters not implemented or incorrectly parsed.
  - Pagination headers (e.g., `Link`) missing or malformed.
  - Temporal queries with `temporalProperty` or `timerel` not implemented.
- **Impact:** Cross-compatibility for data discovery and historical data retrieval was reduced.

A.3.5 Error Handling and Status Code Usage

- **Problem:** Some systems returned ambiguous or non-standard HTTP status codes when handling invalid requests.
- **Typical causes:**
  - Lack of standardized error response formats.
  - Overuse of HTTP 500 for client-side input errors.
- **Impact:** Difficulties in debugging interoperability issues and limited ability to automate test validation.

A.4 Best Practices

A.4.0 Foreword

The following best practices are derived from lessons learned during the interoperability event and are intended to promote reliable and semantically consistent NGSI-LD implementations. These guidelines should be considered by both platform developers and data providers to ensure correct behavior and compatibility.

A.4.1 Implement Cross-Vendor Subscription Testing

- **Why:** Subscriptions and notifications are critical for real-time systems but are often implemented inconsistently.
- **Practice:**
  - Test subscription creation, triggering, and deletion across multiple broker implementations.
  - Ensure notifications include expected metadata, `@context`, and updated entity fragments.

    - Validate that subscription conditions (e.g., observed attributes, geofencing) are honored.
- **Benefit:** Enhances real-time interoperability and facilitates event-driven use cases.

A.4.2 Follow Consistent Naming and ID Conventions

- **Why:** Uniform naming helps humans and machines understand and process data more effectively.
- **Practice:**
    - Use lowercase for properties and relationships (e.g., `airQuality`, `measuredBy`).
    - Use CamelCase or PascalCase for types (e.g., `AirQualityObserved`, `Device`).
    - Generate URNs for entity IDs using the pattern: `urn:ngsi-ld:<EntityType>:<ID>`
- **Benefit:** Simplifies debugging, reduces schema mismatches, and promotes clarity.

A.4.3 Log and Monitor Context Resolution Failures

- **Why:** Many interoperability problems stem from issues resolving `@context` references.
- **Practice:**
    - Add logging for failed context resolution and unexpected JSON-LD terms.
    - Use local copies of remote contexts during offline testing.
- **Benefit:** Identifies semantic misalignments early and helps diagnose source of context parsing failures.

A.4.4 Normalize Timestamps and Use ISO 8601 Format

- **Why:** Time-based operations (e.g., temporal queries, subscriptions) depend on consistent time representations.
- **Practice:**
    - Use full ISO 8601 timestamps with UTC (`YYYY-MM-DDTHH:MM:SSZ`) in all `observedAt`, `createdAt`, `modifiedAt` fields.
    - Store timestamps in a canonical format and timezone to avoid misinterpretation.
- **Benefit:** Prevents issues in temporal filtering and time-based triggers.

A.5 Recommendations for Implementers

A.5.0 Foreword

To ensure successful adoption and interoperability of NGSI-LD implementations, developers and integrators should follow the practices outlined below. These recommendations are based on real-world issues observed during the event and reflect a consensus on how to reduce integration friction and improve conformance with the NGSI-LD specification.

A.5.1 Validate Payloads with JSON-LD Expansion

- Use a JSON-LD processor (such as jsonld.js, rdflib, or pyld) to expand and normalize payloads.
- Test payloads both before sending (producers) and upon receiving (consumers) to ensure semantic consistency.
- Automate validation as part of CI/CD pipelines.

A.5.2 Implement Context-Aware Entity Creation and Update

- Ensure entities can be created and updated using compacted payloads that include a `@context` array.
- Handle cases where incoming contexts are remote, relative, or custom-defined.
- Validate that the `@context` provides full and unambiguous definitions for all terms used.

A.5.3 Ensure Compatibility with Query Mechanisms

- Support filtering mechanisms as defined in the API spec:
  - `q`, `georel`, `geometry`, `coords`, `idPattern`, etc.
- Implement proper parsing of query parameters, including multiple filters in a single request.
- Support advanced queries like:
  - Query by Relationship (e.g., `entities that belongTo X`)
  - Spatial queries (e.g., entities within a polygon)
  - Temporal queries (e.g., entities updated after `t1`)

A.5.4 Implement Proper HTTP Error Handling

- Follow the HTTP semantics and NGSI-LD error format for all error cases.
- Always return:
  - `400 Bad Request` for malformed input
  - `404 Not Found` for missing entities or resources
  - `422 Unprocessable Entity` for semantically invalid requests
- Include a `type`, `title`, and `detail` in the response body using `application/problem+json`.

A.5.5 Support Pagination and Result Metadata

- Implement pagination support for `GET /entities`, `GET /temporal/entities`, etc., using the `limit` and `offset` parameters.
- Include proper HTTP headers:
  - `Link` for next/prev pages
  - `NGSILD-Results-Count` for total result size
  - `Via` to avoid loops with federated requests
- Provide consistent and predictable ordering of results.

A.5.6 Document Implementation-Specific Behaviors

- Clearly document:
  - Which features are supported (e.g., temporal support, geoqueries)
  - Any known limitations or deviations from the NGSI-LD spec
  - Versioning policies for entities and context models
- Make OpenAPI documentation or Postman collections publicly available if possible.

A.5.7 Participate in ISG CIM Activities

- Participate to the ISG CIM API weekly calls to be updated to the newest changes in the NGSI-LD standard.
- Engage in pre-event interoperability check sessions, when offered, to verify basic compatibility with other implementations.
- Test using official or community-maintained test tools, simulators, or reference implementations (e.g., FIWARE Orion-LD, Scorpio Broker).
- Run test suites provided by ETSI or the event organizers to ensure readiness before formal testing.

History

Document history		
<Version>	<Date>	<Milestone>

Latest changes made on 2025-04-24