भारतीय मानक
Indian Standard

IS 18003 (Part 2) : 2021

यूनिफाइड डाटा एक्सचेंज

भाग 2 ए पी आई स्पेसिफिकेशंस

Unified Data Exchange

Part 2 API Specifications

ICS 33.020, 35.020

भारतीय मानक ब्यूरो
BUREAU OF INDIAN STANDARDS
मानक भवन, 9 बहादुरशाह ज़फर मार्ग, नई दिल्ली – 110002
MANAK BHAVAN, 9 BAHADUR SHAH ZAFAR MARG
NEW DELHI-110002
www.bis.gov.in www.standardsbis.in

June 2021 **Price Group 14**

FOREWORD

This Indian Standard (Part 2) was adopted by the Bureau of Indian Standards, after the draft finalized by the Smart Infrastructure Sectional Committee, had been approved by the Electronics and Information Technology Division Council.

This standard is one of the series of Indian Standards on Unified data exchange. Other standards published so far in the series are:

Part 1 Architecture

The Composition of the panel, LITD 28/P12 and the sectional committee, LITD 28 responsible for the formulation of this standard is given at **Annex F**.

Contents

Pages

Pages

Pages

Pages

LIST OF FIGURES

LIST OF TABLES

Pages

Pages

0 INTRODUCTION

The next wave of smart cities intends to use data-driven innovative solutions to overcome the challenges of urbanization. Harnessing the value of enormous data generated by cities today can solve some of the key challenges faced by the cities. The current smart city implementations are unable to satisfy this need efficiently, due to the proprietary and ad-hoc nature of the interfaces and their implementations. This leads to data exchange bottlenecks thereby making it difficult to develop next generation data driven solutions, such as solutions based on the Artificial Intelligence/Machine learning (AI/ML) technologies, for providing new solutions and services at scale. The Data Exchange (DX) layer, which is an integral part of the Data Layer, as specified in IS 18002 : 2021 aims to address this gap by providing a standardized framework for accessing data in a unified format, allowing for authorized sharing of data between different entities, such as various departments in a city or between various public and private data providers and third party application developers etc. The seamless exchange of data is envisioned to lead to the development of innovative, data based solutions as well as provide an opportunity for data providers and application developers to participate in an urban data marketplace.

This Standard defines Unified Data Exchange interface specifications. It defines a set of APIs that enables controlled and secure any-to-any exchange of all forms of public and privately owned *non-personal* data between data providers and consumers.

Standardized APIs help build robust application ecosystems that not only improve development cycle times but also lead to improvement in reusability and extensibility of the developed applications. With this objective, this Standard defines APIs for interactions with the Data Exchange layer. The interfaces are described in terms of HTTP protocol bindings.

Indian Standard

UNIFIED DATA EXCHANGE

PART 2: API SPECIFICATIONS

1 SCOPE

This Indian Standard (Part 2) defines the API specifications for the Data exchange interfaces identified in the Data exchange reference architecture described in Part 1 of this standard. The API specifications are defined for usage over HTTP protocol only.

The target audience for this standard (Part 2) is the community of software developers who may be the implementers of the Data exchange layer services or may be the users of the Data exchange layer, e.g., Data exchange data publishers or Data exchange data consumers wishing to write applications using data available with the Data exchange.

2 REFERENCES

The standards given below contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of these standards.

IS 18003 (Part 1) : 2020	Unified Data Exchange Framework Part 1 Architecture
IS 18002 (Part 1) : 2021	Unified Digital Infrastructure — Data Layer Part 1 Reference Architecture
IETF RFC 7231	"Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content". Available at https://tools.ietf.org/html/rfc7231
IETF RFC 7232	"Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests". Available at https://tools.ietf.org/html/rfc7232.
IETF RFC 3986	"Uniform Resource Identifier (URI): Generic Syntax". Available at https://tools.ietf.org/html/rfc3986.
IETF RFC 8259	"The JavaScript Object Notation (JSON) Data Interchange Format". Available at https://tools.ietf.org/html/rfc8259.
IETF RFC 7946	"The GeoJSON Format". Available at https://tools.ietf.org/html/rfc7946.
IETF RFC 8141	"Uniform Resource Names (URNs)". Available at https://tools.ietf.org/html/rfc8141.
OGC 06-103r4	"OpenGIS® Implementation Standard for Geographic information — Simple feature access — Part 1: Common architecture". Available at https://portal.opengeospatial.org/files/?artifact_id=25355.
JSON-LD 1.1	A JSON based serialization for linked data. W3C Recommendation, July 2020. Available at: https://www.w3.org/TR/json-ld11/
IETF RFC 7807	Problem Details for HTTP APIs. Available at: https://tools.ietf.org/html/rfc7807
ISO 8601-1 : 2019	Date and time — Representations for information interchange — Part 1: Basic rules Available at https://www.iso.org/standard/70907.html
ISO 8601-2 : 2019	Date and time — Representations for information interchange — Part 2: Extensions Available at https://www.iso.org/standard/70908.html
IETF RFC 2818	"HTTP Over TLS". Available at https://tools.ietf.org/html/rfc2818.
IETF RFC 5246	"The Transport Layer Security (TLS) Protocol Version 1.2". Available at https://tools.ietf.org/html/rfc5246.
ETSI GS CIM 009 V1.4.1 (2021-02)	Context Information Management (CIM); NGSI-LD API. Available at: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.01_60/gs_CIM009v010401p.pdf.

3 TERMINOLOGY AND ABBREVIATIONS

3.1 Terminology

For the purpose of this standard, the definitions given in Table 1 shall apply.

Table 1 Definition of key terms used in this document

Term	Definition
Provider	**Legal Entity:** Human (possibly delegated by an Organization), Organization or an organizational role that has responsibility to provide authorization to use resources.
Consumer	**Legal Entity:** Human or Group or Organization or an organizational Role that consumes a resource via a web application.
Data Exchange	**Service:** Hosts and manages meta-data about data resources, manages authorization for accessing the resources and provides data access for the available data resources.
DX Catalogue service	**Service**: Provides services to manage meta-information about data resources and provides search functionalities to discover data resources hosted with the data exchange. A software entity providing this service will be referred to as Catalogue Server.
DX Resource Access Service	**Service**: Serves resources to authorized Apps/Consumers. A software entity providing this service will be referred to as Resource Server.
DX Authorization Service	**Service**: Provides authorization to access data for data resources in accordance to the access policies set for the resources. A software entity providing this service will be referred to as Authorization Server.
Authorization Token	A digital entity that is used to present the authorization credentials to the DX Catalogue and Resource Server.
Authentication Token	A digital entity used to prove the identity of a user to the DX Authorization Service.
Catalogue Item	An item/document in the DX Catalogue meta-information store that describes the meta-information associated with DX entities, such as a data resource, a group of data resources, a DX provider etc. Information contained in a catalogue item depends upon the type of the item.
App	**Application**: Software (like a mobile app, web app, device app or server app), that uses resources to provide a service or experience to the Consumer.
ProviderApp	**Application**: An App that enables a Provider to manage the meta-data and access control in the data exchange, for the resources they are responsible for.

3.2 Abbreviations

Table 2 List of bbreviations used in this document

Abbreviation	Definition
DX	Data Exchange
JSON	JavaScript Object Notation
API	Application Programming Interface
RS	Resource Server
CS	Catalogue Server
AS	Authorization Server
TLS	Transport Layer Security
CRUD	Create, Read, Update, Delete API operations
JSON-LD	JavaScript Object Notation for Linked Data
JWT	JSON Web Token
URN	Uniform Resource Name
URL	Uniform Resource Locator
IRI	Internationalized Resource Identifier
AMQP	Advanced Message Queuing Protocol
MQTT	Message Queuing Telemetry Transport
JMS	Java Message Service
UUID	Universally Unique IDentifier
XML	eXtensible Markup Language
IdP	Identity Provider Service

4 DATA EXCHANGE API INTERFACES

The high level architecture of the data exchange (DX) is presented in **IS 18003 (Part 1)**. The DX reference architecture recognizes two key stakeholders i.e. Data providers and Data consumers. A data consumer consumes a data resource via a client application for application development. A data provider is the provider of the data resources and has the responsibility to provide authorization for the use of its data resources. The key objective for the DX is to enable seamless, secure access of data to the consumers while respecting access control policies set by the provider. Towards that the DX defines a set of interfaces as shown in **Fig. 1** of **IS 18003 (Part 1)** (reproduced here for ease of reference) and described in **Table 2** of **IS 18003 (Part 1)**.

This standard provides detailed specifications for some of the above interfaces. In particular, the Discovery (D), Management (M), Authorization (A) and Resource Access (R) interfaces are defined as part of this standard. As mentioned in **IS 18003 (Part 1)**, defining

the Consent (C) and Identity (I) interfaces is out of the scope of the DX specifications.

The services view of DX reference architecture organizes the above interfaces into the following set of services:

a) **Catalogue Service:** A catalogue server hosts different types of meta-information for the data resources available with the data exchange. The interfaces exposed by the catalogue service are as follows:
   1) Search / Discovery (D) - Allows applications to search and discover items.
   2) Management (M) - Allows provider applications to manage the item's metadata.

b) **Authorization Service:** An authorization server provides authorization to access data according to the access policies set by the provider using the following interfaces:
   1) Authorization (A) - Allows consumers to get authorization tokens and resource servers to validate the authorization tokens.
   2) Resource Management (A) - Allows providers to set and control the resource access policies.

c) **Resource Access Service:** A resource server serves the data to consumers using the following interfaces:
   1) Resource Access (R) - Allows consumers to access data.

Tables 3-5 summarize the functionality of each service using a set of API endpoints exposed by each service.

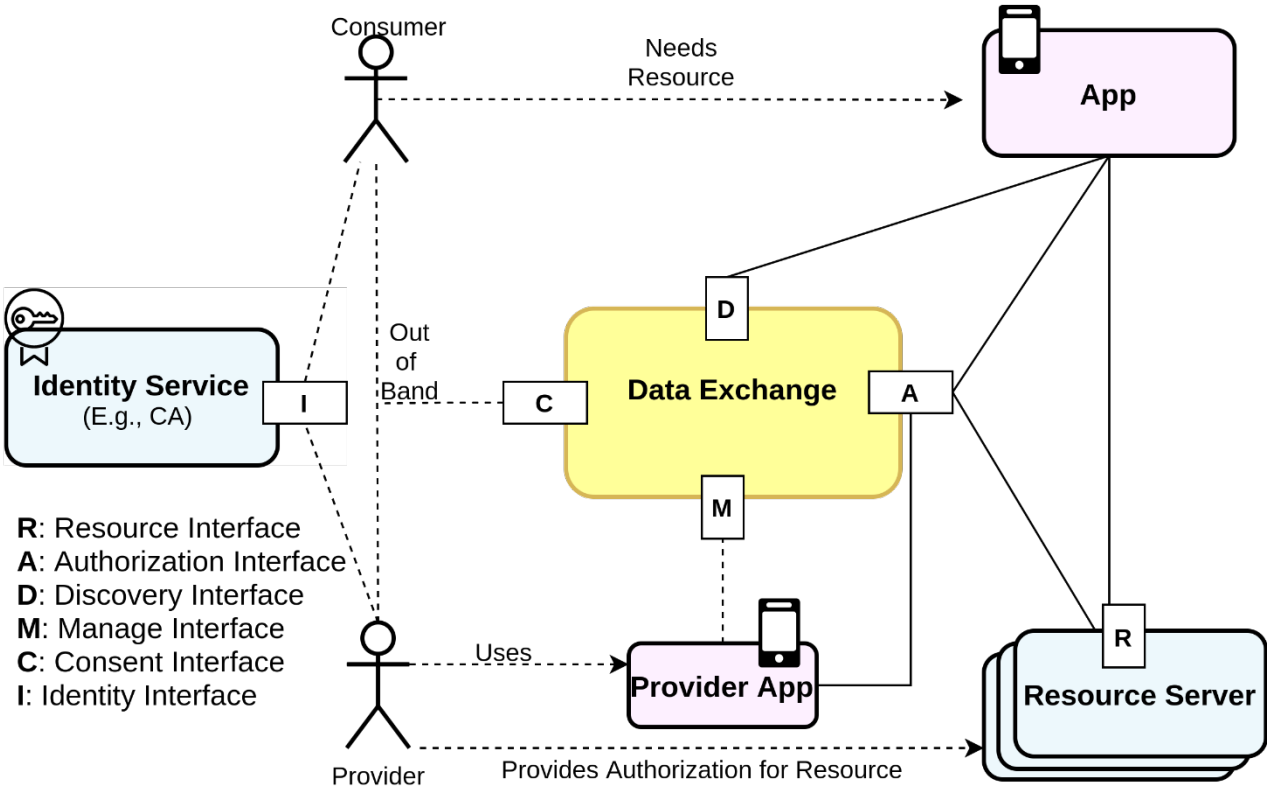

FIG. 1 DATA EXCHANGE REFERENCE ARCHITECTURE

Table 3 Data Exchange Interfaces and APIs for Catalogue Service

Interface	API Endpoint	Description
Manage (M)	/items	Create, update and manage the meta-information of resources in the catalogue.
Discover (D)	/search	Search the catalogue for resources and groups (GET or POST with search queries only).
	/list{entityType}	List all the entities of a particular type.

Table 4 Data Exchange Interfaces and APIs for Resource Access Service

Interface	API Endpoint	Description
Resource (R)	/entities/{ID}	Search operation to retrieve latest data of a resource (or) a resourceGroup.
	/entities	Search and Count operations to retrieve archived data for a resource using spatial, attribute queries and filters.
	/temporal/entities	Search and Count operations to retrieve archived data for a resource using temporal, spatial, attribute queries and filters.
	/entityOperations/query	Search and Count operations to retrieve archived data for a resource using spatial, attribute queries and filters using a POST query.
	/temporal/entityOperations/query	Search and Count operations to retrieve archived data for a resource using temporal, spatial, attribute queries and filters using a POST query.
	/subscription	CRUD operations to subscribe for a resource. This will allow the consumers to get data as a stream (*See* Section **6.1.3.5**)

Table 5 Data Exchange Interfaces and APIs for Authorization Service

Interface	API Endpoint	Description
Authorization (A)	/policies	CRUD operations on policies by an authenticated provider, and read operation by authenticated consumers. The users can only access/modify policies which they are associated with.
	/tokens	CRUD operations on authorization tokens by authenticated consumers.
	/user/profile	CRUD operations on user profiles by authenticated users. All users can access/ modify only the details associated with their profile. The profile details are used for user verification and security processes.

The API specifications in this standard are defined for usage over HTTP protocol, as described in IETF RFC 7231 and IETF RFC 7232, only. Further, the current specifications mandate that all the APIs shall use TLS protection over its API endpoints. All APIs specified in this standard shall support HTTP over TLS (*see* IETF RFC 2818, IETF RFC 5246).

Clause **5** to **7** describe the above APIs in detail. In particular, the Methods, Status codes, Query Parameters, Responses and Functionalities for each API are defined. Common behaviours applicable to all APIs defined in this standard are described in **8**. **Annex D** provides illustrative DX API usage examples and **Annex E** provides a few illustrative use cases of the data exchange layer.

5 CATALOGUE SERVICE INTERFACE

5.1 Catalogue Information Model

5.1.1 General

A catalogue is a collection of items that describe various types of meta-information associated with the data resources available with the DX. For example, a data descriptor to describe data available with a resource, provider information, resource description, discovery hints etc. The information contained in an item is represented via a set of ***attributes*** along with their ***values***. Catalogue information model describes the structure of this meta-information: types of catalogue items, the relationship between the item

types, attributes contained in each item type, attribute data formats etc.

5.1.2 Catalogue Items and Item Types

A DX Catalogue consists of an unordered collection of 'items'. Each item consists of a set of 'attribute' and 'value' pairs. Each item shall contain the following two attributes:

a) 'id': To associate a unique identifier for a catalogue item
b) 'type': To associated a 'type' with a given catalogue item

An item 'type' serves to categorize the type of information represented by a catalogue item. Each 'type' defines a *mandatory* set of attributes that the item belonging to that type shall contain.

An item may contain additional meta-information attributes. Catalogue server implementations are free to specify additional meta-information attributes for a given 'type' and/or additional item types to provide enhanced functionality.

A 'type' can also be associated with a complex object which may appear as a 'value' object for another attribute.

In general, a 'type' serves to define a 'schema' of a catalogue item or a complex object contained within an item. Extensible meta-information content imposes a known syntactic structure that enables defining a minimal set of search and discovery operations yet allowing extensions in features provided by individual DX catalogue server implementations.

A Catalogue server implementation shall specify the schemas for the various item types. The schema definitions shall include an unambiguous description of expected attributes and a specification of a complete set of mandatory attributes. It is recommended that formal frameworks, such as JSON Schema, JSON-LD 1.1 etc., be used for this purpose.

DX catalogue defines the following base set of item types:

a) 'Resource',
b) 'ResourceGroup',
c) 'ResourceServer',
d) 'Provider'.

A 'Resource' item contains meta-information about an individual resource for which data is available via the DX Resource Server.

A 'ResourceGroup' item contains meta-information pertaining to a group of resources. A resource group defines a grouping of resources with the following characteristics:

a) All resources in a group are available with the same Resource Server,
b) All resources shall belong to the same 'Provider' entity, and
c) A given resource can belong to at most one resource group.

Resource groups help to organize the available resources in an efficient manner, e.g., multiple air quality monitoring sensors from the same provider can be grouped together or multiple resources related to transit management may be grouped together etc.

A 'Provider' item contains information about the DX provider entity, such as name of the 'Provider', description, contact information etc.

A 'ResourceServer' item contains information about 'Resource Server' that is hosting a set of data resources available on the data exchange, such as identity, description, name, domain names etc.

The 'id' in a catalogue item serves as a unique DX system-wide identifier for the entity it represents. For example, the catalogue id for a 'Resource' item has to be used to access data from this resource on a Resource server or to set policies for this resource on the Authorization server.

DX catalogue may also contain an item or an object of type '**DataDescriptor**'. An object of type 'DataDescriptor' describes various data attributes available with the data for a given resource. The description of a data attribute is via an extensible set of meta-information attributes and may include data formats (strings, number etc.), allowed range of values, units in case of observed quantities, measurement resolutions and accuracies etc. Standardization of this set of attributes is out of the scope of this standard and is left as an implementation choice. It is recommended that a DataDescriptor may provide linkages with known vocabularies and data models[1)], thereby enabling the consuming applications to gain a better semantic understanding of data.

A 'DataDescriptor' object may be present in a DX catalogue as an item of type 'DataDescriptor' or as an object of type 'DataDescriptor' contained as a value of attribute 'dataDescriptor' or as an external object referenceable through a URL link provided in the 'dataDescriptor' attribute.

[1)] Such as schema.org. smartdatamodels.org

Description of various attributes associated with the above item types is presented in Clause **5.1.5**.

5.1.3 Relationship between catalogue item types

'Resource' and 'ResourceGroup' are the two key items around which catalogue services have been designed. These two items consolidate all types of meta-information related to the data resources served by the DX.

A 'Resource' represents the most atomic level at which access to the data and authorizations can be provided. A resource is always associated with a resource group and can belong to at most one resource group. Resources within a resource group are necessarily hosted on the same resource-server and belong to the same provider entity. The concept of resource group enables certain operations, e.g., authorization grants, data descriptor definitions, search and discovery etc., to be defined on a group level. Another advantage of such a grouping is that it enables an efficient organization of meta-information by collating all the common information across associated resources in a single object. Thus, resources within a resource server are organized as mutually disjoint groups.

Fig. 2 summarizes the relationship between various DX catalogue objects.

A 'Resource' entity shall always contain a reference to the associated 'ResourceGroup'. Additionally, a 'resource' item may contain a reference to 'Provider' and/or 'ResourseServer' item also.

A 'ResourceGroup' shall contain a reference to the 'Provider' item corresponding to the provider associated with the resource group.

A 'ResourceGroup' item shall contain a reference to a 'ResourceServer' item corresponding to the associated Resource Server that hosts the 'ResourceGroup' and the associated 'Resources'.

Either the 'ResourceGroup' or the 'Resource' item shall contain either the 'DataDescriptor' object or a link to a 'DataDescriptor' object. In general, if the data descriptor is common to all the resources in the resource group then it may be included in the resource group. However, if all the resources do not have a common data descriptor then the data descriptor object or its link should be contained in each resource item.

The references to another catalogue item and/or to another object having an Internationalized Resource Identifier (IRI) are represented using attributes with 'Relationship' data type.

NOTES

1 Relationship attributes allow one to link various catalogue items (and even external objects) and view the catalogue as a graph with catalogue items as nodes and relationship attributes as edges. This enables the relationship queries on the catalogue (as described in **5.2.2.4**) which may be extended to graph queries in future.

2 Only the provider entity associated with 'ResourceGroup' and 'Resources' is allowed to create/modify/delete these items.

5.1.4 Catalogue data types

Table 6 lists various data types used to describe data formats for various meta-information attributes defined in the schema tables.

Table 6 Data Types Used in Catalogue

Attribute Data Type	Description
String	Text for string values.
Number	Number for representing numeric values (Decimals, Integers, etc.).
DateTime	Describes ISO Time in a DateTime format as specified in ISO 8601.
Relationship	Class of all Relationships between items in data exchange. A relationship property relates a resource to another resource. The range of relationship properties are objects describing the links (URIs etc.) to the related resource and may contain meta-information about the type of relationship.
GeoJSONGeometry	A Geo-JSON Object as per the GeoJSON format as described in IETF RFC 7946.
Object	A generic object.

5.1.5 Schemas for DX catalogue item types

This section provides a list of attributes/properties associated with different DX catalogue item types.

Tables 7-10 describes the properties associated with an item of types 'Resource', 'ResourceGroup', 'Provider' and 'ResourceServer'. Properties with '*' indicates a mandatory property.

A catalogue service implementation should ensure that all the mandatory attributes are included in a catalogue item of a given type.

In addition to the above described mandatory attributes, a catalogue implementation may define an additional set of mandatory or optional attributes. A catalogue implementation may choose to specify additional item types and their associated attributes. The descriptions of those additional attributes and item types are left to the individual catalogue implementation.

5.1.6 Representation of catalogue entities

Catalogue entities shall be represented using JavaScript Object Notation (JSON) format, as described in IETF RFC 8259. . JSON is an open data interchange

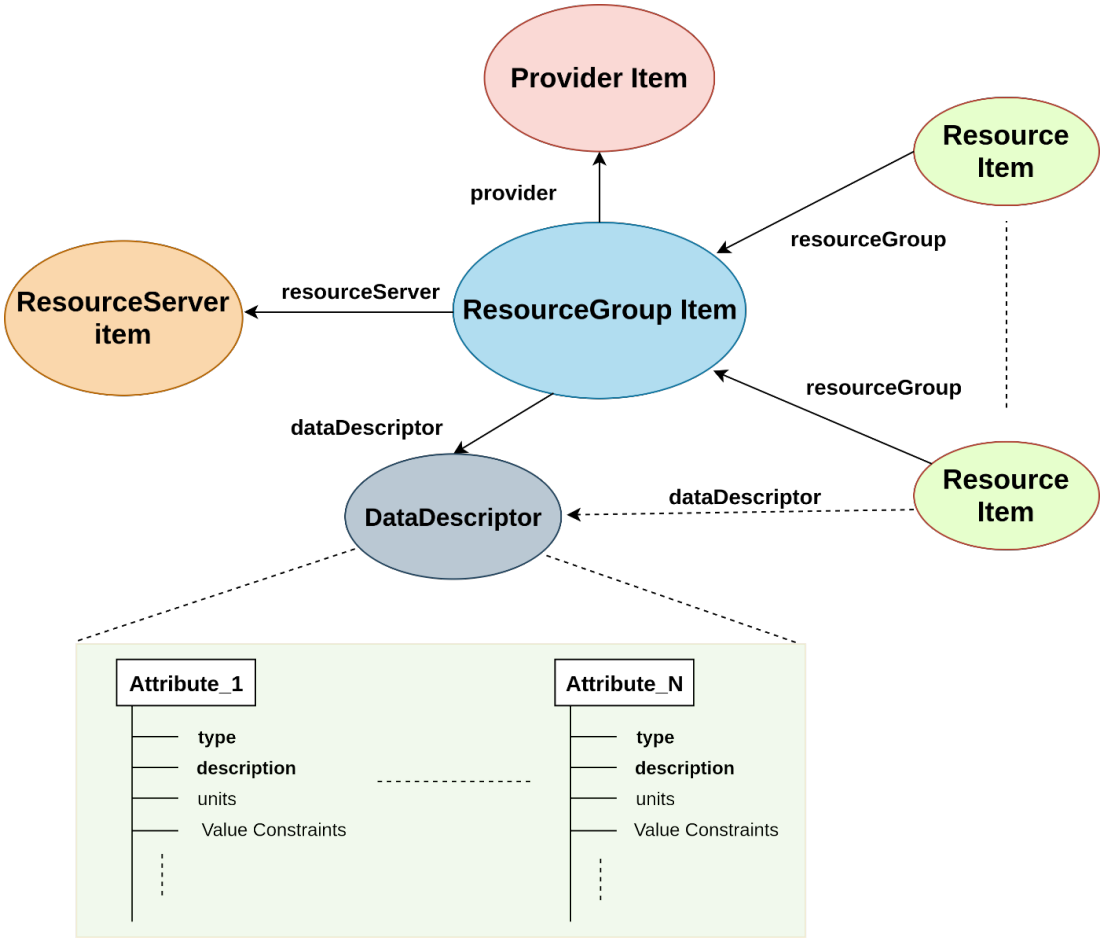

FIG. 2 RELATIONSHIP BETWEEN VARIOUS CATALOGUE ITEMS

Table 7 Properties of Iem of type 'Resource'

Attribute name	Attribute Type	Attribute Description
id*	String	Identifier for the catalogue item.
type*	String	Type of catalogue item.
tags*	String	Comma separated discovery tags associated with an DXEntity (resource , resource Group).
description*	String	Textual description for an DX entity
resourceGroup*	String	DX Catalogue id of the ResourceGroup entity associated with the resource.
provider*	String	DX Catalogue id of the Provider entity associated with the given catalogue item
name*	String	Name of the entity.
dataDescriptor	Object	Descriptor object describing the resource data entities as mentioned in **5.1.2**.
accessPolicy	String	Defines the access policy for the resource. It can take values: [OPEN, SECURE]. OPEN implies the access to the Resource is open (does not need any authorization token for data access). SECURE implies the access to the Resource is not open (it will need an authorization token for data access).
location	String/Object/ GeoJSONGeometry	Location associated with a resource Item (a spatial point), a group of resource items (a spatial region).

Table 8 Properties of item of type 'ResourceGroup'

Attribute name	Attribute Type	Attribute Description
id*	String	Identifier for the catalogue item.
type*	String	Type of catalogue item.
tags*	String	Comma separated discovery tags associated with an DXEntity (resource , resource Group)
name*	String	Name of the entity.
description*	String	Textual description for the catalogue entity.
resourceServer*	String/Object	DX Catalogue id of the ResourceServer entity for a given Resource or ResourceGroup.
provider*	String/Object	DX Catalogue id of the Provider entity associated with the given catalogue item
resourceType*	String	Type of resource. ENUM: [MESSAGESTREAM, DATASET, FILE, MEDIASTREAM, MESSAGE].
accessPolicy*	String	Defines the access policy for resources belonging to ResourceGroup or an individual Resource. In the case of a ResourceGroup item, it can take values: [OPEN, SECURE, MIXED]. OPEN implies all Resources in the group are open (and need no authorization token for data access). SECURE implies that all Resources are secured and will need authorization token for access. MIXED implies that some of the Resource items may be open. In case of accessPolicy being set to MIXED in the resource group, the individual resource items will further qualify the access policy using this field.
dataDescriptor	Object	Descriptor object describing the resource data entities as mentioned in **5.1.2**.
location	String/Object/ GeoJSONGeometry	Location associated with a resource Item (a spatial point), a group of resource items (a spatial region).

Table 9 Properties of item of type 'Provider'

Attribute name	Attribute Type	Attribute Description
id*	String	Identifier for the catalogue item.
type*	String	Type of catalogue item.
name*	String	Name of the entity.
description*	String	Description of the DX catalogue item.
providerOrg*	Object	Information about the Provider Organization.
location	String/Object/ GeoJSONGeometry	Location (or a coverage region) associated with the Provider entity.

Table 10 Properties of item of type 'ResourceServer'

Attribute name	Attribute Type	Attribute Description
id*	String	Identifier for the catalogue item.
type*	String	Type of catalogue item.
name*	String	Name of the entity.
description*	String	Description of the DX catalogue item.
location	String/Object	Location (or a coverage region) associated with a Resource Server.
resourceServerHTTPAccessURL	String	Resource Server URL for data access based on HTTP/REST APIs
resourceServerStreamingAccessURL	String	Resource Server URL for data access based on Streaming Protocols
resourceServerOrg	Object	Information about the organization operating the Resource Server.

format that uses human readable text to represent data objects consisting of attribute-value pairs. The catalogue information model described above lends itself naturally to JSON representation. Further, it makes it easier for catalogue service implementations to describe the structure unambiguously using schema frameworks such as JSON schema.

An implementation may choose advanced JSON-based formats, such as JSON-LD, as specified in W3C JSON-LD 1.1, to represent catalogue entities. Any such formats used to represent catalogue entities shall be compliant to JSON format, i.e., catalogue items shall be valid JSON documents.

Examples of catalogue items in JSON format are presented in **Annex A**.

5.2 DX Catalogue APIs

5.2.1 General

The catalogue service provides meta information about data resources registered with the DX system and facilitates their discovery by DX data consumers and their management by the DX data providers. This section describes APIs for discovery and management of various catalogue entities which form a part of the catalogue information model as described in **5.1**. The APIs are described in terms of HTTP structure, using methods, query parameters, filters, request and response bodies.

In terms of functionality, the APIs exposed by DX Catalogue service can be grouped into the categories of *Discovery* and *Management*.

5.2.1.1 *Discovery interface*

The catalogue discovery interface allows DX consumers to discover data resources of interest and find meta-information related to these resources. The following search and discovery functionalities are supported by DX catalogue service:

5.2.1.1.1 *Attribute/Property Search*

It allows users to discover entities with matching values for certain attributes contained within catalogue items. It is essentially a key-value pair search. For example, one can find all resource group items using property search on item type attribute.

5.2.1.1.2 *Geo-spatial search*

It allows users to discover data resources using geo-spatial search. For example, it enables users to locate data resources such as sensors, cameras etc., in a particular spatial region.

5.2.1.1.3 *Text search*

It allows users to find entities using inexact string matches with the content of a select set of text valued attributes in catalogue items such as descriptions, names, tags etc.

5.2.1.1.4 *Get Document by ID*

It allows the users to get the exact catalogue entity by providing the *id* of the entity.

5.2.1.1.5 *List based on type*

It allows users to list all documents of a given *type* in the catalogue. The type may be *ResourceGroup, ResourceServer* or *Provider*.

5.2.1.1.6 *Relationship search based on relationship type*

It allows users to find all entity documents which are related to the root document through the specified relationship.

5.2.1.2 *Management Interface*

The Management Interface allows users to create, update and delete meta-information documents (items) in the catalogue. The following management functionalities are supported by DX catalogue service:

5.2.1.2.1 *Create Item*

Create / Index an item of a given type, as mentioned in **5.1**, in the catalogue.

5.2.1.2.2 *Update Item*

Update an already existing item in the catalogue.

5.2.1.2.3 *Delete Item*

Delete an already existing item in the catalogue.

5.2.1.2.4 *Get Item*

Get a document from the catalogue given its *id*.

5.2.2 API Endpoints

This section describes various API endpoints exposed by the DX catalogue service. For details on query semantics and API parameters refer to **5.2.4**. For details on response body objects refer to **5.2.5**.

5.2.2.1 *API Endpoint: /item*

The /item APIs are a set of DX Provider oriented APIs to index (create) and manage (update/delete) Catalogue documents/items.

Tables 11-14 describe the API parameters, methods and status codes as applicable for different functionalities for the endpoint /item.

Table 11 Parameters and Status codes for Create item

Functionality	Create item
Methods	POST
Required post body	The request body contains the catalogue item to be created. The schemas for different catalogue item types, e.g., Resource, Resource Group etc., are defined in **5.1**. The value of attribute 'type' in the request body defines the type of catalogue entity that is to be created and the request body should adhere to the schema specified for the corresponding catalogue entity type.
Status codes	201, 400, 401

Table 12 Parameters and Status codes for Update item

Functionality	Update item
Methods	PUT
Required put body	The request body contains updated properties of the catalogue item to be updated with the already existing id of the document. The schemas for different catalogue item types, e.g., Resource, Resource Group etc., are defined in **5.1**. The value of attribute 'type' in the request body defines the type of catalogue entity that is to be created and the request body should adhere to the schema specified for the corresponding catalogue entity type.
Status codes	200, 400, 401, 404

Table 13 Parameters and Status codes for Delete item

Functionality	Delete item
Methods	DELETE
Required query parameter	id
Status codes	200, 400, 404

Table 14 Parameters and Status codes for Get item

Functionality	Get item
Methods	GET
Required query Parameter	id
Status codes	200, 400, 404

5.2.2.1.1 *API Response*

Table 15 describes different response status codes, responses and scenarios under which the response occurs for the /search API.

Table 15 Response codes for API endpoint /item

HTTP Status code	Response Body Format	Scenario
200	DX-CAT-Success-Response-Search	Successfully found document
	DX-CAT-Success-Response-Create-Update-Delete	Successfully deleted document
201	DX-CAT-Success-Response-Create-Update-Delete	Successful creation of document
		Successful update of document
400	DX-CAT-Error-Response	Invalid Syntax, Invalid param type
401	DX-CAT-Error-Response	Missing Token, Invalid Token, Token expired etc. (Refer **11.1**)
404	DX-CAT-Error-Response	No such document exists

5.2.2.2 *API Endpoint: /search*

The /search API allows DX consumers to discover metadata documents using key-value (property), geo-spatial and fuzzy text query parameters. Further a combination of the three can be used to issue more complex queries. The query semantics for various search functionalities are described in **5.2.4**.

Tables 16-18 describe the API parameters, methods and status codes as applicable for the different search functionalities.

Table 16 Parameters and Status codes for Property Search

Functionality	Property search (*see* **5.2.4.1**)
Methods	GET
Required query parameters	property, value
Optional query parameter	limit, offset, filter
Status codes	200, 400

Table 17 Parameters and Status codes for Geo-spatial Search

Functionality	Geo-spatial search (*see* **5.2.4.2**)
Methods	GET
Required query parameters	georel, geometry, maxDistance, coordinates
Optional query parameters	geoproperty, limit, offset, filter
Status codes	200, 400

Table 18 Parameters and Status codes for Fuzzy Text Search

Functionality	Fuzzy Text search (*see* **5.2.4.3**)
Methods	GET
Required query parameters	q
Optional query parameters	limit, offset, filter
Status codes	200, 400

5.2.2.2.1 *API Response*

Table 19 describes the different response status codes, responses and scenarios under which the response occurs for the /search API.

Table 19 Response codes for API endpoint /search

HTTP Status code	Response Body Format	Scenario
200	DX-CAT-Success-Response-Search	Successful Query
400	DX-CAT-Error-Response	Invalid Syntax, Invalid param type

5.2.2.3 *API Endpoint: /list*

The /list API allows DX consumers to list entity document id's of a given type. This is useful in enumerating all documents of a certain type for consumption of user interfaces. The query semantics for various search functionalities are described in **5.2.4**.

Table 20 describes the API parameters, methods and status codes as applicable to the endpoint /list.

Table 20 Parameters and Status codes for List item

Functionality	List based on type (*see* **5.2.4.5**)
Methods	GET
Required path parameter	type
Status codes	200, 400

5.2.2.3.1 *API Response*

Table 21 captures the different response status codes, responses and scenarios under which the response occurs for the /list API.

Table 21 Response codes for API endpoint/list

HTTP Status code	Response Body Format	Scenario
200	DX-CAT-Success-Response-List	Successful Query
400	DX-CAT-Error-Response	Invalid type

5.2.2.4 *API Endpoint: /relationship*

The /relationship API allows DX consumers to discover metadata documents which are related to a root entity document through some relationship.

Table 22 Parameters and Status codes for Relationship search

Functionality	Relationship search based on relationship type (*see* **5.2.4.6**)
Methods	GET
Required query parameters	id, rel
Status codes	200, 400

5.2.2.4.1 *API Response*

Table 23 describes the different response status codes, responses and scenarios under which the response occurs for the /list API.

Table 23 Response codes for API endpoint / relationship

HTTP Status code	Response Body Format	Scenario
200	DX-CAT-Success-Response-Search	Successful Query
400	DX-CAT-Error-Response	Invalid type

5.2.3 API Authorization

Catalogue management APIs that require creation, updation and deletion of meta-information objects require authorization. The catalogue server shall support token based authorization. Authorization mechanisms should ensure that only authorized users are allowed to create, update and delete catalogue items. Token generation, token validation and token formats are excluded from the scope of this standard. A 'Resource', 'ResourceGroup' and 'Provider' items shall only be created/updated/deleted by the associated provider role or a role acting on behalf of the provider. 'ResourceServer' item may be created/updated/deleted by a DX administrator.

5.2.4 Query Semantics and API Parameters

5.2.4.1 *Property Search Semantics and Parameters*

The property search finds all catalogue entities where the value of a given property/attribute matches exactly to the value provided in the query input. The property search is applicable to string valued properties only, i.e., properties whose values are strings or an array of strings. One may combine multiple properties in a search through a logical-AND ("&&"). Further, for

each such property multiple matching values may be given. Multiple values for a given property are matched via a logical-OR ("||") operation.

The parameters for a property query are:

a) **property** *(Array[String]):* Array of properties (keys) on which query is to be made. The mapping between a property and a value is one-to-one

b) **value** *(Array[Array[String]]):* Applicable values of the one-to-one mapped properties.

The syntax for a property query is as follows: **property**=[<prop-1>,<prop-2>]&**value**=[[<prop-1-val-1>,<prop-1-val-2>],[<prop-2-val-1>]]. This search will return entities with the following matching rule: '{ <prop-1> ==<prop-1-val-1> || <prop-1> ==<prop-1-val-2> } && { <prop-2> ==<prop-2-val-2>}'. For property search only exact string matches are applicable. Property search can be applied to any property and may include nested properties through **prop.subprop.subsubprop** syntax, wherever applicable. Note that an implementation may choose to expose only a limited set of attributes/properties for a given entity type for this search. In which case such information should be made available by the respective implementations.

Typically, this search is used to find entities with required values for properties such as id, resourceGroup, provider, resourceServer, type etc. Some common uses are:

a) search discovery tags: property=[tags]&value=[["aqm","pollution"]]

b) search by type of entity: property=[type]&value=[[ResourceGroup]]. Valid entities are Resource, ResourceGroup, Provider, ResourceServer.

c) search resources belonging to a group: propert y=[resourceGroup]&value=[[<group-id>]]

5.2.4.2 *Geo-spatial Search Query Semantics and Parameters*

The Geo search is to search entities using geo-spatial attributes. A geo-spatial query intends to find all catalogue entities with geo-spatial bindings (specified using parameter **geoproperty**) that satisfies a specific spatial relation (specified using parameter **georel**), with the query input geometry (specified using parameter **geometry** and **coordinates**).

The geometrical data types used in the geo-spatial query are based on GeoJSON geometry definitions as described in Section 3.1 of IETF RFC 7946. In particular, geometry type (e.g., point, polygon etc.) and coordinate objects representations are based on GeoJSON specification.

The parameters used in geo-spatial query are as follows:

a) **geoproperty** *(String):* This parameter specifies the geo-spatial attribute present in the catalogue entities that defines the target geometry. This parameter is optional and a given catalogue service implementation may specify a default value, such as 'location', to be used for this parameter.

b) **geometry** *(String):* This parameter specifies the type of input geometries for the spatial query. This parameter takes on one of the following values from the list of GeoJSON geometry types (*see* IETF RFC 7946): **Point**, **Polygon**, **LineString**, **Bbox**. Along with the **coordinates** parameter, this parameter completely specifies the input geometry.

The value of 'Point' is included to represent a circle shape (which does not have a direct representation in GeoJSON) where the specified coordinates of 'Point' geometry represent the center of the circle. It shall always be accompanied with a query parameter **maxDistance** which represents the radius of the circle.

c) **georel** *(String):* This parameter specifies the spatial relationship to be used in the spatial query along with the input query geometry. This parameter should take one of the following values: ['within', 'intersects', 'disjoint']. A brief description of these spatial relations connecting a target geometry with input query geometry is as follows:

1) **within**: Finds all entities whose target geometries are within the close shaped input query geometry.

2) **intersects**: Finds all the entities whose target geometries intersects with the query input geometry.

3) **disjoint**: Finds all entities whose target geometries are disjoint to, i.e. falling outside, the close shaped input query geometry.

In the above spatial relationships **within** and **disjoint** are not applicable when input geometry type is **Linestring**.

d) **maxDistance** *(Integer):* This parameter defines the radius in meters when the input geo shape is a circle. It is applicable only when the geometry parameter equals **Point**.

e) **coordinates** *(Array[Array[Array[Double]]] or Array[Array[Double]] or Array[Double]):* This parameter specifies the coordinates to represent the input geometry. The data format is the same as the corresponding GeoJSON

'coordinate' object as specified in IETF RFC 7946. Depending upon the geometry type this parameter accepts string encoded one dimensional and multidimensional arrays. For example, [longitude-1,latitude-1] should be used for the geometry **Point**. [[[longitude-1,latitude-1], [longitude-2,latitude-2],...., [longitude-n,latitude-n], [longitude-1,latitude-1]]] should be used for geometry **Polygon**. [[longitude-1,latitude-1], [longitude-2,latitude-2],...., [longitude-n,latitude-n]] should be used for geometry **Linestring.** Finally, [[longitude-1,latitude-1], [longitude-2,latitude-2]] should be used for geometry **bbox**. Note that, for polygon coordinates the first coordinate is the same as the last. For bbox, the two coordinate points represent the top-left and bottom-right vertices of the bounding box.

A given implementation may choose to specify additional constraints on allowed values of coordinate objects for the input geometry, e.g., number of coordinates in a polygon/linestring etc., in which case this information should be made available to the DX consumers via appropriate documentation.

5.2.4.3 *Fuzzy Text Search Semantics and Parameters*

A fuzzy text search intends to find catalogue entities containing non-exact or similar matches to a target text pattern provided as a query input. It searches across all entity types.

a) **q** *(String):* The pattern to be in-exactly matched is provided using parameter

The set of properties/attributes that are part of fuzzy text search is implementation dependent. Further, the metrics used to evaluate 'similarity' or 'in-exactness of match', e.g., Levenshtein distance, are also dependent on a given implementation and it is recommended that this information should be made available by the catalogue service implementations.

5.2.4.4 *Complex Search Semantics and Parameters*

A combination of Property Search, Geo-spatial Search and Text Search parameters constitutes a complex search

5.2.4.5 *List Entities Semantics and Parameters*

List all entities in the DX catalogue based on the ***type*** of the catalogue entity. The parameters used in list entities query are as follows:

a) **type** *(String):* The 'type' of the entities whose list is to be obtained. This is a string enumeration whose values should be one of the following: [resourceGroup, resourceServer, provider]

5.2.4.6 *Relationship Query Semantics and Parameters*

Entity documents in the catalogue are related to each other with "relationships" as shown in Figure 2. The relationship is represented by an attribute whose value is the DX catalogue ID for another entity. For example, a resource item will contain an attribute 'resourceGroup' whose value is the DX catalogue id for the Resource Group to which this resource belongs. It is possible to define a query semantic to obtain related entities given the **id** of the root entity and the relationship for which the related entities are required. Such a query will be very useful for example to fetch all resource items belonging to a given resource group.

The parameters accepted by this query are as follows:

a) **id** *(String)*: id of the root entity

b) **rel** *(String):* Relationship name for which entities related to the root entity are required. Current specifications support the following values for **rel**: [provider, resourceGroup, resourceServer, resource].

Because of the hierarchical nature of entities in the catalogue, only certain root-entity types-relationship combinations are allowed and their responses will be either a single document or a list of documents. Table 24 shows all supported combinations:

Table 24 Supported combinations for entity types and relationships in relationship search

Type of the entity (id)	Supported relationships for "rel"	Description
Resource	resourceGroup,resourceServer, provider	Return the associated resourceGroup (or resourceServer or provider) entity. Only a single entity is returned.
ResourceGroup	resource	Return all resource items associated with the input resource group. Multiple entities may be returned.
ResourceGroup	resourceServer, provider	Return the associated resourceServer (or provider) entity. Only a single entity is returned.
Provider	resource, resourceGroup	Return all resource or resource group items associated with the input provider. Multiple entities may be returned.
ResourceServer	resource, resourceGroup	Return all resource or resource group items associated with the input resource server. Multiple entities may be returned.

5.2.4.7 *Limits and Filters*

Discovery endpoint supports additional optional parameters to limit and filter the results of the query. In particular, the following two operations are supported:

a) Attribute filtering: Only a subset of properties of returned documents can be requested in the response. The attributes of interest can be specified using the parameter **filter**. For e.g, **filter**=[id,name] returns documents with only the id and name property retained in the returned documents.

b) Pagination: Catalogue service APIs shall support query pagination. Query pagination support is defined using parameters: **limit** and **offset**. For details refer to Section 8.4.

5.2.5 DX-CAT Object Definitions

This section contains some common object definitions used within request/response bodies for the DX Catalogue service. All the API responses follow the template defined in **8.1**. Extending the responses mentioned there, the following section describes some specific responses for the different API endpoints and functionalities.

5.2.5.1 *DX-CAT-Success-Response-Search*

The response format (and/or attributes and attribute values) for a successful search query shall be as given in Table 25.

Table 25 Catalogue search response attributes

Attribute Name	Attribute Data Type	Attribute Value
type	String	A well defined URN detailing the response type.
title	String	A human readable title for the response.
results	Array[JSON Objects]	A JSON Array containing the matched entities of the query.
totalHits	Number	Total number of entities matching the request.

5.2.5.2 *DX-CAT-Success-Response-List*

The response format (and/or attributes and attribute values) for a successful list operation shall be as given in Table 26.

5.2.5.3 *DX-CAT-Success-Response-Create-Update-Delete*

The response format (and/or attributes and attribute values) for a successful create/update/delete shall be as given in Table 27.

Table 26 Catalogue list response attributes

Attribute Name	Attribute Data Type	Attribute Value
type	String	A URN indicating a successful request.
title	String	A human readable title for the successful response.
results	Array[JSON Objects]	A JSON Array containing a list of **id**'s of the matched entities.
totalHits	Number	Total number of entities matching the request.

Table 27 Catalogue Create, Update, Delete response attributes

Attribute Name	Attribute Data Type	Attribute Value
type	String	A well defined URN detailing the response type.
title	String	A human readable title for the response
results	Array[JSON Objects]	A JSON Array containing a list of objects whose schema is as shown in Table 28.
totalHits	Number	Total number of entities matching the request

5.2.5.3.1 *DX-CAT-Manage-Response*

Table 28 Catalogue management API response attributes

Attribute Name	Attribute Data Type	Attribute Value
id	String	Generated id in the case of Create Item or Document id in the case of update and delete.
method	String	HTTP method used on this API endpoint (Post/Put/Delete).
status	String	Status of the request. May indicate a successful request or a pending for approval.

6 RESOURCE ACCESS SERVICE INTERFACE

6.1 Resource Access APIs

6.1.1 General

A resource access service provides data access for a given data resource using search and subscription APIs which are specified in this section. A resource is identified using the 'id' field from the corresponding 'Resource' entity in the DX catalogue. A 'Resource' represents an individually access-controlled and queryable data source in a DX system. This section defines various data access APIs available for a given

resource. The APIs are defined for usage over HTTP protocol and are described using methods, query parameters, filters, request and response bodies.

The resource access APIs use token-based authorization to allow a consumer application to access data for a given resource. A consumer application can obtain an 'authorization token' using the token grant service provided by the DX Authorization server. The consumer application then passes the 'authorization token', using the "token" field in the header, as the credential for the requested data access API. Based on the authorization information contained in the token, as specified in **7.1.2**, a resource server should accept or deny the data access request. Note that it is the responsibility of the Resource Access Service to enforce the access permissions communicated via the authorization tokens.

The authorization tokens, which are issued by DX Authorization server, can be in any format, the definitions of which are out of scope of the current specifications. It is recommended that the DX Authorization service implementation choose standard token formats such as JWT access tokens as described in IETF RFC 7519, OAuth2.0 bearer tokens as described in IETF RFC 6750 etc. It may be noted that, for opaque/proprietary token formats, Resource Access service implementation can use the Token Introspection (TIP) service provided by the DX Authorization server to decode a given token.

> NOTE — the authorization token may contain additional optional resource specific authorization information to allow for advanced access controls for a given resource. Although, the definition of these additional access control attributes is out of the scope of this specification, it shall be noted that the definition of the attributes and the access control implications implied by these attributes should be mutually agreed upon between Resource Server and DX Authorization Server implementations.

The current resource access service specifications are aligned with the NGSI-LD API Specifications as described in ETSI GS CIM 009 V1.4.1 (2021-02). Currently, the scope of alignment is limited to API endpoints, functionality and query parameters. Towards this alignment some features from the current draft specifications were also included in the latest versions of NGSI-LD API specifications.

6.1.2 Functionality

The Resource Access Service provides the following functionality with which users can query a data source in a DX system. As mentioned in **6.1.1**, in terms of functionality the current resource access service specifications are fully aligned with the NGSI-LD API Specifications.

6.1.2.1 *Latest Data Search*

The Latest Data search allows users to get the latest (last published) data of a resource.

6.1.2.2 *Temporal Search*

Temporal search allows users to get data of a resource using time property based queries. It intends to find all the data where temporal properties satisfy given temporal constraints.

6.1.2.3 *Spatial Search*

Spatial search allows users to get data of a resource using a geo-spatial query. A geo-spatial query, can be specified using parameters specifying geo property, geo relationship, geometry and coordinates, as specified in **6.1.4**. It intends to find all the data where the input spatial relationship exists between the input geometry and the geometry specified by the geo property attribute of the data for a given resource.

6.1.2.4 *Attribute Search*

Attribute search is used to get data of a resource using a comparison operator which performs a specific mathematical, relational or logical operation. An attribute query returns all documents which matches the query-specified operation.

6.1.2.5 *Complex Search*

Complex search is used to get data of a resource using temporal, spatial and attribute queries. A complex query returns all documents which matches the query-specified time, area and operation.

6.1.2.6 *Filters*

Using filters users can request the resource access service to respond with a subset of properties in the matched documents. It can be used along with any of the search functionalities provided in **6.1.2.1** to **6.1.2.5**.

6.1.2.7 *Subscription*

Subscription allows users to access resources as a stream using streaming protocols such as MQTT as defined in MQTT 5.0, AMQP as defined in AMQP 0.9.1, Apache Kafka, Java Message Service (JMS) etc. By registering a subscription with the resource access service, users shall be provided with a dedicated channel with which data will be made available.

6.1.2.8 *Data Ingestion*

As mentioned in the reference architecture, **IS 18003 (Part 1)**, the DX does not mandate how the data is ingested into the resource access service. It is left to the implementations to choose the most efficient means of collecting data from various sources such as edge devices, sensor clouds, vendor-specific APIs etc. In case a resource server chooses to provide a publish/ingestion service to data providers to onboard their data, this document makes some non-normative

recommendations about ingestion/publish APIs. These recommendations are presented in **Annex B**.

6.1.3 API Endpoints

This section describes various API endpoints exposed by the DX resource access service. For details on query semantics, parameters refer to **6.1.4**. For details on response templates and status codes, refer to **6.1.5**.

6.1.3.1 *API Endpoint: /entities*

The /entities API allows DX consumers to query a data resource using spatial and attribute queries using resource id, spatial and attribute parameters. It allows consumers to construct a geo-spatial search, attribute search and a complex search (combination of geo-spatial search and attribute search). Further, by passing the resource id as a path parameter, the API endpoint can be used to get the latest data for a resource.

Along with a geo-spatial search, attribute search and a complex search (combination of geo-spatial search and attribute search), if an options query parameter with value equal to count is passed, the/entities API will respond with a count of records instead of actual data.

If the data resource is a protected resource a valid authorization token is mandatory in the header parameters of the request. Information about the resource id of the resource and security scope of the resource can be obtained from the associated meta-information in the DX catalogue**.**

Tables 29-31 summarize the applicable parameters for each search functionality.

Table 29 Parameters and Status codes for latest data search

Functionality	Latest Data search
Methods	GET
Required path parameters	id
Status codes	200, 204, 400, 401, 404

Table 30 Parameters and Status codes for geo-spatial search

Functionality	Geo-spatial search (*see* **6.1.4.1**)
Methods	GET
Required query parameters	id, georel, geometry, coordinates
Optional query parameters	geoproperty, attrs, options, limit, offset
Status codes	200, 400, 401, 404

Table 31 Parameters and Status codes for attribute search

Functionality	Attribute Search (*see* **6.1.4.3**)
Methods	GET
Required query parameters	id, q
Optional query parameters	attrs, options, limit, offset
Status codes	200, 400, 401, 404

Note that this endpoint also supports a complex search which is a combination of geo-spatial and attribute search. It combines individual searches using logical AND operation (*see* **6.1.4.4**) and the parameter set for the complex search is the union of parameter sets for all the geo-spatial and attribute searches.

6.1.3.1.1 *API Response*

Table 32 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

Table 32 Response codes for Resource Access Service API endpoint /entities

HTTP Status code	Response Body Format	Scenarios
200	DX-RS-Success-Response-Search	Successful Query
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Authorization Token, Invalid Authorization Token, Authorization Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX

6.1.3.2 *API Endpoint: /temporal/entities*

The /temporal/entities API allows DX consumers to query a data resource using temporal, spatial and attribute queries using resource id, temporal, spatial and attribute parameters. It allows consumers to construct a temporal search and a complex search (using combinations of temporal, spatial and attribute searches).

Along with a temporal search, and a complex search, if an **options** query parameter with value equal to **count** is passed, the /temporal/entities API will respond with count of data records found instead of actual data.

If the data resource is a protected resource a valid authorization token is mandatory in the header parameters of the request. Information about the

resource id of the resource and security scope of the resource can be obtained from the associated meta-information in the DX catalogue**.**

Table 33 summarizes the applicable parameters for the temporal search functionality.

Table 33 Parameters and Status codes for Temporal search

Functionality	Temporal search (*see* **6.1.4.2**)
Methods	GET
Required query parameters	id, timerel, time, endtime
Optional query parameters	timeproperty, attrs, options, limit, offset
Status codes	200, 400, 401, 404

NOTES

1 this endpoint also supports a complex search. The following three combinations are allowed: a combination of temporal and geo-spatial (type-1), or temporal and attribute search (type-2) or temporal and geo-spatial and attribute search (type-3). It combines individual searches using logical AND operation as explained in **6.1.4.4**. The parameter set for the complex searches is the union of parameter sets for all the involved individual searches.

2 in order to use this API, temporal query parameters are mandatory.

6.1.3.2.1 *API Response*

Table 34 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

Table 34 Response codes for Resource Access Service API endpoint/temporal/entities

HTTP Status code	Response Body Format	Scenario
200	DX-RS-Success-Response-Search	Successful Query
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Authorization Token, Invalid Authorization Token, Authorization Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX

6.1.3.3 *API Endpoint: /entityOperations/query*

The /entityOperations/query API allows DX consumers to query a data resource using spatial and attribute queries using POST method where geo-spatial and attribute search parameters are passed using a request body. The semantics of the query itself is exactly the same as the GET query described in /entities endpoint above including complex search and count operations.

The reason behind defining a POST query to perform a similar operation as defined in **6.1.3.1** is to get around possible limitations on supported query string sizes by different browsers/applications.

If the data resource is a protected resource a valid authorization token is mandatory in the header parameters of the request. Information about the resource id of the resource and the security scope of the resource can be obtained from the associated meta-information in the DX catalogue**.**

The request body schema for this query is defined by entity DX-RS-ReqBody-Entities described in **6.1.5**.

Table 35 summarizes the applicable parameters for the POST based search functionality.

Table 35 Parameters and Status codes for POST based search

Functionality	Geo-spatial search and Attribute Search (*see* **6.1.4**)
Methods	POST
Required body parameters	type, entities, geoQ (required for spatial and complex), q (required for attribute and complex)
Optional body parameters	attrs, options
Optional query parameters	limit, offset
Status codes	200, 400, 401, 404

NOTE — the body parameter geoQ is a JSON object that further contains geo-spatial search parameters: geometry, georel and geoproperty.

6.1.3.3.1 *API Response*

Table 36 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

Table 36 Response codes for Resource Access Service API endpoint /entityOperations/query

HTTP Status code	Response Body Format	Scenario
200	DX-RS-Success-Response-Search	Successful Query
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Authorization Token, Invalid Authorization Token, Authorization Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX

6.1.3.4 *API Endpoint: /temporal/entityOperations/query*

The /temporal/entityOperations/query API allows DX consumers to query a data resource using temporal, spatial and attribute queries using POST method where temporal, geo-spatial and attribute search parameters are passed using a request body. The semantics of the query itself is exactly the same as the GET query described in /temporal/entities endpoint above including complex search combinations and count operations.

As described above, the reason behind defining a POST query to perform a similar operation as defined in **6.1.3.2** is to get around possible limitations on supported query string sizes by different browsers/ applications.

If the data resource is a protected resource a valid authorization token is mandatory in the header parameters of the request. Information about the resource id of the resource and the security scope of the resource can be obtained from the associated meta-information in the DX catalogue**.**

The request body schema for this query is defined by entity DX-RS-ReqBody-Search described in **6.1.5**.

Table 37 summarizes the applicable parameters for POST based temporal and complex search functionality.

Table 37 Parameters and Status codes for POST based temporal search

Functionality	Temporal search and Complex Search (*see* **6.1.4**)
Methods	POST
Required body parameters	type, entities, temporalQ, geoQ (required only for complex search), q (required only for complex search)
Optional body parameters	attrs, options
Optional query parameters	limit, offset
Status codes	200, 400, 401, 404

NOTES

1 the body parameter temporalQ is a JSON object that further contains temporal search parameters: timerel, time, endtime and timeproperty.

2 in order to use this API, temporal query parameters are mandatory.

6.1.3.4.1 *API Response*

Table 38 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

Table 38 Response codes for API endpoint / temporal/entityOperations/query

HTTP Status code	Response Body Format	Scenario
200	DX-RS-Success-Response-Search	Successful Query
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Authorization Token, Invalid Authorization Token, Authorization Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX

6.1.3.5 *API Endpoint: /subscriptions*

The /subscriptions API allows DX consumers to register, modify, list and delete a subscription for one or more data resources through a streaming service, such as MQTT, JMS, Apache Kafka, AMQP etc. The specific streaming service supported is implementation dependent and is out of scope of the current specification.

The subscription API shall be a protected API. A valid authorization token is mandatory in the header parameters of the request. Information about the resource id of the resource and security scope of the resource can be obtained from the associated meta-information in the DX catalogue.

For future extensibility, to allow for subscription modes other than streaming, such as call backs, this API shall require a header parameter **options**. For the current specifications, which only support streaming mode of subscription, this parameter shall always be set equal to ‘streaming’.

The request body schema for this API is defined by entity DX-RS-ReqBody-Subscription described in **6.1.5**.

A DX consumer may use the POST method to register a subscription, PUT method to replace the resources (modify) in an existing subscription and PATCH method to append resources to the existing subscription.

Table 39-40 summarizes the applicable parameters for subscription functionality.

Table 39 Parameters and Status codes for subscription registration, update and append

Functionality	Register / Update (Replace) / Append (add) a set of resources for subscription.
Methods	POST, PUT, PATCH
Required body parameters	name, type, entities
Status codes	200, 400, 401, 404, 409, 201 (only for POST)

Table 40 Parameters and Status codes for listing and deleting resources in subscription

Functionality	List / Delete resource(s) subscribed
Methods	GET, DELETE
Required path parameters	subscriptionID
Status codes	200, 400, 401, 404

6.1.3.5.1 *API parameters*

DX resource access subscription APIs accept the name, type and entities request body parameters as described in **6.1.5.6**.

In addition to the above the following parameters as mentioned in Table 41 are required:

6.1.3.5.2 *API Response*

Table 42 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

6.1.4 Query Semantics and API Parameters

This section describes the parameters used in the resource access service APIs and their roles with respect to the overall API functionality.

6.1.4.1 *Geo-spatial Search*

A geo-spatial query intends to find all the data for a given resource that satisfies a specific spatial relation (specified using parameter **georel**), with the query input geometry (specified using parameter **geometry** and **coordinates**). The data for the resource is expected to contain geo-spatial attributes (specified using parameter **geoproperty**) that specify the target geometry used in the spatial relation evaluation.

The geometrical data types used in the geo-spatial query are based on geoJSON geometry definitions as described in Section 3.1 of IETF RFC 7946. In particular, geometry type (e.g., point, polygon etc.) and coordinate objects representations are based on GeoJSON specification.

A geo-spatial query is constructed using the following parameters:

a) **geoproperty** *(String)*: This parameter specifies the geo-spatial attribute/property that defines the target geometry. This shall match a geo-spatial attribute present in the data for a given resource. This parameter is optional and a given implementation can specify a default value to be used for this parameter.

Table 41 Response codes for Resource Access Service API endpoint /subscription

Parameter Name	Parameter Value Data Type	Parameter Type	Description
subscriptionID	String	Path	Represents the unique subscriptionID for a subscription (See DX-RS-Subscription-Params object for more details.)
options	String	Header	Represents the streaming options requested by the consumer.

Table 42 Resource Access Service /subscription API response codes

HTTP Status code	Response Body Format	Scenario
200	DX-RS-Subscription-Success-Response	Successful create/update/list operation.
	DX-RS-Success-Response-Delete	Successful delete operation.
201	DX-RS-Subscription-Success-Response	Subscription registration successful
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Authorization Token, Invalid Authorization Token, Authorization Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX
409	DX-RS-Error-Response	Resource already registered

b) **georel** *(String)*: This parameter specifies the spatial relationship to be used in the spatial query along with the input query geometry. This parameter should take one of the following values: ['near;maxdistance=<distance_in_meters>', 'within', 'intersects', 'contains', 'equals', 'disjoint', 'overlaps']. A brief description of these spatial relations connecting a target geometry with input query geometry is as follows:

1) ***within***: Target geometry is within the close shaped input query geometry as specified by OGC 06-103r4.
2) ***intersects***: Target geometry intersects with the input query geometry as specified by OGC 06-103r4.
3) ***near***: Target geometry is within a circle shaped input geometry. The circle shape is specified using a center point and radius. The center point is defined using a 'point' valued **geometry** parameter (*see* below). The radius is specified using the 'maxdistance' parameter which shall be specified whenever **georel** has the value 'near'. The value of **maxdistance** is specified in meters.
4) ***contains*:** Target geometry is contained within the close shaped input query geometry as specified by OGC 06-103r4.
5) ***equals*:** Target geometry is equal to the close shaped input query geometry as specified by OGC 06-103r4.
6) ***disjoint*:** Target geometry is disjoint to the close shaped input query geometry as specified by OGC 06-103r4.
7) ***overlaps*:** Target geometry overlaps to the close shaped input query geometry as specified by OGC 06-103r4.

NOTES

1 All fields in the above enum list are case sensitive fields.

2 The relation near;maxdistance is applicable only for a 'point' valued **geometry** parameter (*see* below).

3 An implementation may choose to support only a subset of the above spatial relationships in which case the capability list should be made available by the resource server implementations, e.g., via DX catalogue and/or resource access service documentation etc., to the DX consumers. Additionally, an implementation may specify maximum permissible value for **maxdistance**.

c) **geometry** *(String)*: This parameter specifies the type of input geometries for the spatial queries. This parameter takes the following values from the list of GeoJSON geometry types (*see* IETF RFC 7946): **Point**, **Polygon**, **LineString**, **Bbox**. Along with the **coordinates** parameter, this parameter completely specifies the input query geometry to be used in the spatial query. Note that, the value of 'Point' is included to represent a circle shape (which does not have a direct representation in GeoJSON) where the specified coordinates of 'Point' geometry represent the center of the circle. The parameter **geometry** with value **Point** shall be used only with **georel**: 'near;maxdistance' where the modifier **maxdistance** specifies the radius of the circle in meters.

d) **coordinates** *(Array[Array[Array[Double]]] or (Array[Array[Double]] or Array[Double])*: This parameter specifies the coordinates to represent the input geometry. The data format is the same as the corresponding GeoJSON 'coordinate' object as specified in Section 3.1 of IETF RFC 7946. Depending upon the geometry type this parameter accepts string encoded one dimensional and multidimensional arrays. For example, [longitude-1,latitude-1] should be used for the geometry **Point**. [[[longitude-1,latitude-1], [longitude-2,latitude-2],...., [longitude-n,latitude-n], [longitude-1,latitude-1]]] should be used for geometry **Polygon**. [[longitude-1,latitude-1], [longitude-2,latitude-2],...., [longitude-n,latitude-n]] should be used for geometry **Linestring.** Finally, [[longitude-1,latitude-1], [longitude-2,latitude-2]] should be used for geometry **bbox**. Note that, for polygon coordinates the first coordinate is the same as the last. For bbox, the two coordinate points represent the top-left and bottom-right vertices of the bounding box.

NOTE — a given implementation may choose to specify additional constraints on allowed values of coordinate objects for the input geometry, e.g., number of coordinates in a polygon/linestring etc., in which case this information should be made available to the DX consumers via appropriate documentation.

6.1.4.2 *Temporal Search*

A DX temporal query can be used to query data sets with temporal constraints. In particular, it can be used to request historic values within the specified timeframe.

A temporal search accepts time property based queries which are based on date-time formats specified in ISO 8601 : 2004. The params and their accepted values are as follows:

a) **timeproperty** *(String)*: This parameter specifies the temporal property/attribute to

which the temporal query is to be applied. The value of this parameter shall match a temporal attribute present in the data for a given resource. This parameter is optional and a given implementation may specify a default value to be used for this parameter.

b) **timerel** *(String)*: This parameter specifies the time relation to be used in the temporal query. It specifies the temporal relation that shall exist between the time property in the data and the input temporal parameters. It shall be one of the following: ['between', 'before', 'after']. The interpretations of these values is as described below:
   1) **before**: For a data record to match, the value of the specified temporal property in the data has to be before the time specified by the query parameter **time**.
   2) **after**: For a data record to match, the value of the specified temporal property in the data has to be after the time specified by the query parameter **time**.
   3) **between**: For a data record to match, the value of the specified temporal property in the data has to be between the time specified by the query parameter **time** and **endtime**.

c) **time** *(String<ISO 8601 format>)*: Start time for the temporal query in ISO 8601 format.

d) **endtime** *(String<ISO 8601 format>)*: End time for the temporal query in ISO 8601 format. This parameter is applicable only when **timerel** equals **between**.

NOTE — there may exist implementation dependent limits on the allowed time differences between **time** and **endtime**, in which case the resource access service implementations should make this information available for the consumers, e.g., via DX catalogue or API documentation etc. Similar limits may apply for **before** and **after** temporal queries too.

6.1.4.3 *Attribute Search*

An attribute query works on an attribute level and finds data where the value of the given attribute (target value) satisfies the conditions implied by the query operator and a query input value. An attribute query works with attributes that are "string" and "numeric" valued. For example, the following attribute query 'q=airTemperature>=20' will find data for a given resource where the property airTemperature has value greater than or equal to 20. Table 43 summarizes various operators allowed for the attribute query along with their matching criteria.

6.1.4.4 *Complex Search*

A complex search combines any of the above geo-spatial, attribute or temporal searches. It combines individual searches using logical AND operation. That is, the search output results shall satisfy the matching criteria for each individual search. The parameter set for the complex searches is the union of parameter sets for all the involved individual searches.

6.1.4.5 *Response Filtering*

A resource access query operation can optionally return a result set that can include only the specified attributes. This operation is called filtering. The attributes of interest can be specified in a comma separated format using the query parameter **attrs** *(String)*. If **attrs** is not specified, no filtering is performed and all attributes present in the data shall be returned.

6.1.4.6 *Query pagination*

Resource access APIs shall support query pagination. Query pagination support is defined using parameters: **limit** and **offset**. For details refer **8.4**.

6.1.4.7 *Counting the Number of Results*

DX resource access queries support returning just the number of records found as a result of query execution. This may be useful for consumers in case the number of hits is too large and may require fine tuning of query parameters. This operation is supported via query parameter **options** *(String)*. Setting **options** equal to **count** returns only the number of records matching the query instead of returning data from the query results. This parameter should always be used along with a valid query (spatial, temporal or attribute). Also, note that this parameter should not be used along with the parameter **attrs** (response filtering).

6.1.4.8 *API parameters*

DX resource access service APIs accept the following parameters as described above: **georel**, **geometry**, **coordinates**, **geoproperty**, **timeproperty**, **timerel**, **time**, **endtime**, **q**, **attrs**, **options**, **limit, offset** and **options.**

In addition to above the following parameters as described in Table 44 are required by the DX resource access service APIs:

6.1.5 DX-RS Object definitions

This section contains some common object definitions used within request/response bodies for the DX resource access service. These objects are listed in Tables 45-51.

Table 43 Attribute search query template

Operator	Query Template	Matching Criteria
Equal (represented as ==)	*q=attr1==val1*	Applicable to both string and numeric valued attributes. For string valued attributes: Finds all data for a resource where the value of *attr1* exactly matches val1 or where one of the array elements of *attr1* value matches *val1* exactly, in case *attr1* is an array of strings. For numeric valued attributes: Finds all data for a resource where the value of *attr1* is equal to *val1*.
Not Equal (represented as !=)	*q=attr1!=val1*	Applicable to both string and numeric valued attributes. For string valued attributes: Finds all data for a resource where the value of *attr1* does not match val1 or where none of the array elements of *attr1* value match *val1,* in case *attr1* is an array of strings. For numeric valued attributes: Finds all data for a resource where value of *attr1* is not equal to *val1*.
Greater than (represented as >)	*q=attr1>val1*	Applicable to only numeric valued attributes. Finds all data for a resource where the value of *attr1* is strictly greater than *val1*.
Greater than equal (represented as >=)	*q=attr1>=val1*	Applicable to only numeric valued attributes. Finds all data for a resource where the value of *attr1* is greater than or equal to *val1*.
Lesser than (represented as <)	*q=attr1<val1*	Applicable to only numeric valued attributes. Finds all data for a resource where the value of *attr1* is strictly lesser than *val1*.
Lesser than equal (represented as <=)	*q=attr1<=val1*	Applicable to only numeric valued attributes. Finds all data for a resource where the value of *attr1* is lesser than or equal to *val1*.
Between (represented as ==)	*q=attr1==val1..val2*	For numeric valued attributes: Finds all data for a resource where value of *attr1* lies between *val1* and *val2* with endpoints included, i.e., *val1*<= value(*attr1*) <= *val2*.

NOTE — for a given resource the attribute query may be applicable only to an implementation dependent subset of attributes present in the data. The information about the set of attributes to which this query may apply should be provided to the DX consumer, e.g., through DX catalogue or API documentation etc. Further, an implementation may choose to impose additional restrictions on the allowed values, such as length of input strings, not allowing special characters etc.

Table 44 Resource Access Service API parameters

Parameter Name	Parameter Value Data Type	Description
id	String	Represents the unique ID for the resource as per the DX catalogue.

6.1.5.1 *DX-RS-Success-Response-Search*

Table 45 Response payload schema for successful Resource Access Service search query

Attribute Name	Attribute Data Type	Attribute Value
type	String	A well defined URN as per the response type.
title	String	A human readable title for the response.
results	Array[JSON Objects]	A JSON Array containing data records found as a result of the query. The data attributes contained in returned records depend upon the data Resource. The information about that should be made available by providers of the resource via data descriptor objects contained in the corresponding Resource/ResourceGroup entity in the DX Catalogue.
totalHits	Number	Total number of documents responded for the request.

6.1.5.2 *DX-RS-Success-Response-Delete*

Table 46 Response payload schema for successful Resource Access Service subscription delete operation

Attribute Name	Attribute Data Type	Attribute Value
type	String	A well defined URN as per the response type.
title	String	A human readable title for the response.
results	Array[JSON Objects]	A JSON Array containing the result of the operation.

6.1.5.3 *DX-RS-Success-Response-Subscription*

Table 47 Response payload schema for successful Resource Access Service subscription create/modify operation

Attribute Name	Attribute Data Type	Attribute Value
type	String	A well defined URN as per the response type.
title	String	A human readable title for the response.
results	Array[JSON Objects]	A JSON array containing objects of type DX-RS-Subscription-Params as defined above.

6.1.5.4 *DX-RS Error-Response*

Table 48 Response payload schema for Resource Access Service error response

Attribute Name	Attribute Data Type	Attribute Value
type	String	A well defined URN as per the response type. Note that in this case this value may provide additional hints about the nature of the error response.
title	String	A human readable title for the response.
detail	String	A detailed information regarding the reason for the error.

6.1.5.5 *DX-RS-ReqBody-Search*

Table 49 describes the attributes used in the request body for the POST query described by endpoint / entityOperations.

Table 49 Request payload schema for API endpoint /entityOperations

Attribute Name	Attribute Data Type	Attribute Value
type	String	Should be always equal to **Query**
entities	Array of JSON Objects	Each object contains the **id** of a resource. The semantics of the parameters in the JSON object are as per the id parameter defined in Section 6.1.4.8.
geoQ	JSON Object	Contains attributes for geo-spatial search. That is, **georel**, **geometry** and the optional parameter **geoproperty**. The semantics and formats of the parameters are as per the Geo-spatial Search query parameters defined in Section 6.1.4.1.
temporalQ	JSON Object	Contains attributes for temporal search. That is, **timerel**, **time, endtime** and **timeproperty**. The semantics of the parameters in the JSON object are as per the temporal Search query parameters defined in Section 6.1.4.2. This attribute is applicable with /temporal/entityOperations/query endpoint only.
q	string	The semantics of the parameters are as per the Attribute Search defined in Section 6.1.4.3.
attrs	string	The semantics of the parameters are as per the definition in Section 6.1.4.5.
options	string	The semantics of the parameters are as per the definition in Section 6.1.4.7.

6.1.5.6 *DX-RS-ReqBody-Subscription*

Table 50 Request payload schema for API endpoint /subscription

Attribute	Attribute Data Type	Attribute Value
name	String	A compact alias name for the subscription
type	String	The value of type should always be equal to “subscription” for this API endpoint.
entities	Array[String]	Array of resource id(s) as per DX catalogue.

6.1.5.7 *DX-RS-Subscription-Params*

Table 51 Request payload schema for update operation for endpoint /subscription

Attribute	Attribute Data Type	Attribute Value
subscriptionID	String	Represents the unique subscriptionID for a subscription.
entities	Array[String]	Array of resource id(s) associated with a subscription.

NOTES

1 The Resource Access Service shall ensure that subscriptionID is unique for all subscriptions issued by it. The format of subscriptionID is left out of scope of this specification and is dependent on a given implementation.

2 The above object may contain additional subscription parameters, such as subscription credentials, streaming server URLs, subscription queues etc., which are dependent on the specific streaming service implemented by the resource access service. Details of these specific parameters shall be made available via implementation specific API documentation for a DX resource server.

7 AUTHORIZATION SERVICE INTERFACE

7.1 Authorization Service APIs

7.1.1 General

The Authorization Server (AS) provides Authorization and Accounting services to the users of the DX layer. The main APIs exposed by this interface are grouped into the following categories:

a) User Profile

b) Policy

c) Authorization Token

The authentication mechanism is considered to be out of scope for this specification. Suggested methods can include OpenID Connect. Unless otherwise specified all the APIs specified by the authorization service interface require authentication by the relevant user. These APIs can be further enhanced/limited based on the user role.

7.1.1.1 *User Profile*

The User Profile APIs are a collection of CRUD operations available to all authenticated users. These allow the user to view and modify their profile details which are used by the DX platform to provide any ancillary services such as role management, contact information, etc. A user can access/modify only their own profile.

7.1.1.2 *Policy*

The Policy APIs are a collection of CRUD operations available to DX Providers to manage the access control to their resources. The policy entity is a simple mapping of a resource ID or a resource group ID, and a DX Consumer. Additionally, it may include any relevant scope information required by a DX Resource Server to fulfil the request. Additional policy constraints may be defined at a later date which enables advanced policy enforcements.

7.1.1.3 *Authorization Token*

The Authorization Token APIs are a collection of CRUD operations available to all DX Consumers to obtain and manage the authorization tokens which are accepted by the DX interface services. The authorization tokens are requested by an authenticated consumer by specifying the resources (groups) that need to be associated with the new token. The AS shall ensure that all the relevant policies associated with the requested resources are satisfied before granting the token. This token, when presented at a DX interface shall be introspected (may be online/offline depending on the token format specification, e.g., UUID vs JWT), and the resulting access control decision is enforced.

7.1.2 Authorization Service Object Definitions

This section contains some common object definitions used within request/response bodies for the DX Authorization service. These objects are listed in Tables 52-61.

7.1.2.1 *DX-AS-UserProfile-Entity*

The UserProfile entity attributes are listed in Table 52.

Table 52 UserProfile entity attributes

Attribute Name	Attribute Type	Description
user_id	String	A unique auto generated ID by the Authorization Service identifying the DX user. Shall not be modified after creation.
roles	Array of Strings	An array of valid roles (consumer, provider) that the user wishes to register with.

7.1.2.2 *DX-AS-Policy-Entity*

The Policy entity attributes are listed in Table 53.

Table 53 Policy entity attributes

Attribute Name	Attribute Type	Description
policy_id	String	A unique ID identifying a policy. Shall not be modified after policy creation.
item_id	String	The catalogue ID of the protected item to which the policy applies.
item_type	String	The catalogue type of the protected item (Resource or ResourceGroup)
user_id	String	The user ID to whom this policy applies.
provider_id	String	The user ID of the provider who created the policy. Shall not be modified after policy creation.
policy_expiry	DateTime	The expiry time associated with a policy in an ISO 8601 compatible string.
constraints	JSON Object	A key-value map of <strings, object> which specify optional constraints which are implementation specific

7.1.2.3 *DX-AS-Token-Entity*

The authorization token issued by Authorization service is associated with meta-information containing the attributes listed in Table 54.

Table 54 Token entity attributes

Attribute Name	Attribute Type	Description
token_id	String	A unique ID identifying the token. Shall not be modified after token creation.
expiry	DateTime	The expiry time of the token in an ISO 8601 compatible string.
access_token	String	The access token which is passed to the DX interface, such as RS, by a DX consumer.
status	enum String	The status of a token with possible values as: “active” or “revoked”
server	URL String	A valid DX interface service URL for which this token has been issued.
resources	Array of resource ID Strings	An array of Resource IDs as specified in the DX catalogue.

NOTES

1 Access_token format is not specified in this document. The implementation may choose the formats from available options such as JWT, UUID or any proprietary format. For opaque formats, such as UUID, the AS implementation shall provide a token introspection service to provide the above meta-information associated with the authorization token.

2 AS implementations may choose to add additional meta-information attributes.

7.1.2.4 *DX-AS-TIP-Req*

Table 55 Request payload schema for Token introspection

Attribute Name	Attribute Type	Description
access_token	String	The access_token which is to be introspected. Passed on by a DX consumer to a DX interface while accessing a protected resource.
item_id	String	The resource ID requested by the DX consumer as per the DX catalogue.
item_type	String	The catalogue type of the protected item (Resource or ResourceGroup)

7.1.2.5 *DX-AS-UserProfile-Success-Resp*

Table 56 Response payload schema for successful UserProfile operation

Attribute Name	Attribute Type	Description
type	String	A URN which specifies a machine readable error type.
title	String	A brief human readable title of the error.
detail	String	A detailed human readable error description.
results	JSON Object	JSON object defined by DX-AS-UserProfile-Entity. Response shall contain attributes **user_id** and **roles**.

7.1.2.6 *DX-AS-Policy-Success-Resp*

Table 57 Response payload schema for successful Policy operation

Attribute Name	Attribute Type	Description
type	String	A URN which specifies a machine readable error type.
title	String	A brief human readable title of the error.
detail	String	A detailed human readable error description.
results	JSON Array of Objects	Defined by DX-AS-Policy-Entity. Response shall contain attributes **policy_id**, **item_id**, **item_type**, **user_id** and **provider_id**. May optionally contain the attribute constraints

7.1.2.7 *DX-AS-Token-Success-Resp*

Table 58 Response payload schema for successful Token operation

Attribute Name	Attribute Type	Description
type	String	A URN which specifies a machine readable error type.
title	String	A brief human readable title of the error.
detail	String	A detailed human readable error description.
results	JSON Array of Objects	Defined by DX-AS-Token-Entity. Response shall contain attributes **token_id**, access_token, **status**, **expiry**, **resources**, and **server**.

7.1.2.8 *DX-AS-Error-Response*

Response body format in case of error response for AS API endpoints.

Table 59 Response payload schema for AS Error response

Attribute Name	Attribute Type	Description
type	String	A URN which specifies a machine readable error type. Note that in this case this value may provide additional hints and finer details about the nature of the error.
title	String	A brief human readable title of the error.
detail	String	A detailed human readable error description.

7.1.2.9 *DX-AS-TIP-Success-Response*

Response body format for /tokens/introspect endpoint.

Table 60 Response payload schema for successful Token introspection operation

Attribute Name	Attribute Type	Description
type	String	A URN which specifies a machine readable error type.
title	String	A brief human readable title of the error.
detail	String	A detailed human readable error description.
results	JSON Object	Response shall contain attributes: [**token_id**, **status**, **expiry**, **server**] as defined in DX-AS-Token-Entity; and an array of objects with each object containing [**item_id**, **item_type**, **constraints**] as defined by DX-AS-Policy-Entity.

7.1.2.10 DX-AS-Delete-Success-Response

Table 61 Delete method success response entity attributes

Attribute Name	Attribute Type	Description
type	String	A URN which specifies a machine readable success type.
title	String	A brief human readable title of the success message.

7.1.3 API Specifications

All APIs shall require an authentication token inside a header parameter called “token” unless otherwise specified. The token may be generated by an external Identity Provider (IdP) Service. The mechanics of how the IdP generates this token is considered out of scope for this standard. Common recommended token options include OpenID Connect, SAML as described in IETF RFC 7522, X.509 client certificates and username/password combination. The chosen IdP shall always provide a way to inspect the identity of the user via token introspection.

7.1.3.1 *Endpoint: /user/profile*

This endpoint allows a DX user to manage their user profile. Users can create their user profile as part of the registration process, view their profile and modify their own profile.

7.1.3.1.1 *Create/Update User Profile*

The endpoint /user/profile supports methods POST and PUT to create and update a user profile. To create a user profile a list of roles that the user wishes to register for is required. A user can either be a DX Provider, DX Consumer or both. The API response, after successful registration, shall include an auto generated user ID for the DX user. Many subsequent AS APIs, as mentioned below, require this user ID to identify a given user. The format of the user ID is left open to a given implementation and is out of scope of this standard.

Also note that, additional implementation specific roles may also be allowed by the DX Authorization servers. The implementation may require further information (such as email etc.) for additional services the implementation intends to provide and such information should be made available via appropriate implementation specific API documentation.

A DX user can use the PUT method to modify their own user profile. This method accepts the same input as the POST method. The existing user profile object is replaced with the profile sent as the input of this method.

The request body schema for POST and PUT methods is defined by entity DX-AS-UserProfile-Entity described in **7.1.2**.

Table 62 summarizes the applicable parameters for user profile create functionality.

Table 62 Parameters and Status codes for user profile creation/updation

Functionality	Create/Update user profile
Methods	POST, PUT
Required body parameters	roles
Status codes	201, 400, 401, 409 (only for POST), 200 (only for PUT)

7.1.3.1.1.1 *API Responses*

Table 63 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 63 Response codes for POST/PUT methods for API endpoint /user/profile

HTTP Code	Response Body Format	Scenarios
201	DX-AS-UserProfile-Success-Resp	Successful creation for POST method.
400	DX-AS-Error-Response	Missing information, invalid input, invalid role etc.
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.
409	DX-AS-Error-Response	Profile already exists.
200	DX-AS-UserProfile-Success-Resp	Successful update for PUT method.

7.1.3.1.2 *Read User Profile*

A user can view their user profile by using the GET method. The AS implementation may provide additional query fields and/or filters. Table 64 summarizes the endpoint parameters.

Table 64 Parameters and Status codes for user profile read

Functionality	Read own user profile
Methods	GET
Query parameters	None
Status codes	200, 401

7.1.3.1.2.1 *API Responses*

Table 65 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 65 Response codes for GET method for API endpoint /user/profile

HTTP Code	Response Body Format	Scenario
200	DX-AS-UserProfile-Success-Resp	Successful read
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.

7.1.3.2 *Endpoint: /policies*

This endpoint is used mainly by DX providers to set and manage access control policies to the resources they own. The DX providers can set a policy to grant access to DX consumers. The Read operation can also be used by the DX consumer to view the resources that they have access to. Any additional policies associated with custom roles are left to the discretion of the implementation.

For future extensibility, additional policy objects may be acceptable by a given authorization service implementation. Specifics of new policy expression language, decision rules etc. should be made available for DX users via appropriate documentation.

7.1.3.2.1 *Create/Update Policies*

Policies are created or updated by a DX provider which apply to the resources that they own. These policies are sent as an array of Policy objects. These operations ensure transactional safety by either succeeding for all policies, or in case of an error, aborting without any persisted changes.

The request body schema for POST and PUT methods is defined by entity DX-AS-Policy-Entity described in **7.1.2**.

Table 66 summarizes applicable parameters for policy create functionality.

Table 66 Parameters and Status codes for policy creation/updation

Functionality	Create / Update Policies
Methods	POST, PUT
Required body parameters	policy_id (for PUT only), item_id, item_type, user_id
Optional body parameters	constraints, policy_expiry
Status codes	201, 400, 401, 409, 200 (for PUT only)

7.1.3.2.1.1 *API Responses*

Table 67 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 67 Response codes for POST/PUT methods for API endpoint /policies

HTTP Code	Response Body Format	Scenario
201	DX-AS-Policy-Success-Resp	Successful creation of multiple policies using POST
400	DX-AS-Error-Response	Missing information, invalid input
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.
409	DX-AS-Error-Response	Already exists
200	DX-AS-Policy-Success-Resp	Successful update of multiple policies using PUT

7.1.3.2.2 *Read Policies*

Policies can be viewed by a DX user using the Read operation. This endpoint shall check the authentication token and return only the relevant policies based on the token identity, for example, policies which were either created by that user as a provider, and/or policies which apply to that user as a consumer. The implementation may provide additional search, filtering and pagination options.

Table 68 summarizes the applicable parameters for policy read functionality.

Table 68 Parameters and Status codes for policy read

Functionality	Read Policies
Methods	GET
Required body parameters	NA
Status codes	200, 401

7.1.3.2.2.1 *API Responses*

Table 69 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 69 Response codes for GET method for API endpoint /policies

HTTP Code	Response Body Format	Scenario
200	DX-AS-Policy-Success-Resp	Successful read
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.

***7*.1.3.2.3** *Delete Policies*

Policies may be deleted by the DX provider who had authored the policies. The endpoint accepts an array of policy IDs and it shall ensure transactional safety as specified in the Create operation. When a policy is deleted, all the tokens which were issued against it shall be considered invalidated during token introspection with respect to that particular DX Consumer and Resource combination.

Table 70 summarizes the applicable parameters for policy delete functionality.

Table 70 Parameters and Status codes for policy deletion

Functionality	Delete Policies
Methods	DELETE
Required body parameters	policy_id
Status codes	200, 400, 401

7.1.3.2.3.1 *API Responses*

Table 71 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 71 Response codes for DELETE method for API endpoint /policies

HTTP Code	Response Body Format	Scenario
200	DX-AS-Delete-Success-Response	Successful deletion of multiple policies
400	DX-AS-Error-Response	Missing information, invalid input
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.

7.1.3.3 *Endpoint: /tokens*

This endpoint is used by the DX consumers to create and manage their authorization tokens. The format of the authorization token issued by the authorization service is out of scope for this standard, since it depends on the implementation's non-functional requirements. The metadata associated with the token is specified in the DX-AS-Token-Entity object described in **7.1.2**.

7.1.3.3.1 *Create Method: POST*

Authorization tokens are requested by a DX consumer by providing an individual resource ID in the request body. Note that authorization service implementations may allow multiple resources to be associated with a single token in which case a list of resource IDs may be provided in the request body. All the resources associated with one token shall be served from the same resource server. When a consumer requests for a token against a list of resources, the relevant policies are checked to ensure that the consumer has access to all the resources before issuing the authorization token.

The request body schema for POST method is defined by entity DX-AS-Token-Entity described in **7.1.2**.

Table 72 summarizes the applicable parameters for authorization token grant functionality.

Table 72 Parameters and Status codes for authorization token generation

Functionality	Generate token for a DX consumer
Methods	POST
Required body parameters	resources
Status codes	201, 400, 401

7.1.3.3.1.1. *API Responses*

Table 73 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 73 Response codes for POST method for API endpoint /tokens

HTTP Code	Response Body Format	Scenario
201	DX-AS-Token-Success-Resp	Access token generated successfully.
400	DX-AS-Error-Response	Missing information, invalid input
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token

7.1.3.3.2 *Read Method: GET*

Authorization tokens generated by a DX consumer can be read using this operation for review purposes. It returns the metadata associated with the token as specified in the entity definition (**7.1.2**). The implementation may optionally include linked data such as the relevant policies which were evaluated during the token generation. It may also allow filtering parameters. This method is optional.

7.1.3.3.2.1. *API Responses*

Table 74 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 74 Response codes for READ method for API endpoint /tokens

HTTP Code	Response Body Format	Scenario
200	DX-AS-Token-Success-Resp	Successful read
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.

7.1.3.3.3 *Update Method: PUT*

This method is optional. In case where implementations allow multiple resources per token this method may be used to modify the list of resources associated with a token. This list can be updated dynamically by the consumer to either remove or add new resources to the same token. The list of resource IDs to be updated for an already existing token are passed in the input request body. Multiple authorization tokens can be updated in a single call. Note that all the relevant policies shall be reevaluated before the existing authorization tokens are modified.

The request body for PUT method accepts an array of DX-AS-Token-Entity objects. Each object shall contain two attributes: **token_id** and **resources**, where token_id represents an already existing authorization token to be updated with additional list of resource Ids contained in the **resources** attribute of the same object. The DX-AS-Token-Entity object is described in **7.1.2**.

Table 75 summarizes the applicable parameters for authorization token update functionality.

Table 75 Parameters and Status codes for authorization token updation

Functionality	Generate token for a DX consumer
Methods	PUT
Required body parameters	(token_id, resources) for each DX-AS-Token-Entity object
Status codes	200, 400, 401

7.1.3.3.3.1 *API Responses*

Table 76 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 76 Response codes for PUT method for API endpoint /tokens

HTTP Code	Response Body Format	Scenario
200	DX-AS-Token-Success-Resp	Successful modification of multiple tokens.
400	DX-AS-Error-Response	Missing information, invalid input
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.

7.1.3.3.4 *Delete Method: DELETE*

A DX consumer may revoke authorization tokens which they have generated by providing a list of the token IDs. The operation shall ensure transactional safety, with either all token revocations succeeding or no changes should be applied. Once a token has been revoked, the token introspection call from any DX interface service, such as a resource server, shall fail for that token.

Table 77 summarizes the applicable parameters for authorization token delete functionality.

Table 77 Parameters and Status codes for authorization token deletion

Functionality	Delete authorization tokens for a DX consumer
Methods	DELETE
Required body parameters	Array of token_id. For token_id, *see* DX-AS-Token-Entity object definition.
Status codes	200, 400, 401

7.1.3.3.4.1 *API Responses*

Table 78 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 78 Response codes for DELETE operation for API endpoint /tokens

HTTP Code	Response Body Format	Scenario
200	DX-AS-Delete-Success-Response	Successful deletion of multiple tokens
400	DX-AS-Error-Response	Missing information, invalid input
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.

7.1.3.4 *Endpoint: /tokens/introspect*

This endpoint is used by DX interface service, such as Resource Access Service, to introspect a token generated by DX Authorization service. Its most important use is by Resource Access service to introspect an authorization token presented to it by a DX consumer to access a protected resource. The DX interface shall be authenticated to call this endpoint and can only introspect tokens whose resources are served by that specific interface.

7.1.3.4.1 *Introspect Method: POST*

The introspect operation accepts the authorization token and the resource ID requested by the DX consumer as input parameters. It returns a minimum response of the following mandatory fields: token status, token expiry time and introspection decision (allow/deny). It may also choose to return additional policy related constraints which can be dynamically enforced by the RS.

The request body schema for POST method is defined by entity DX-AS-TIP-Req object described in **7.1.2**.

7.1.3.4.1.1. *API Responses*

Table 79 describes the status codes and response body formats. The response body formats are explained in **7.1.2**.

Table 79 Response codes for POST method for API endpoint /tokens/introspect

HTTP Code	Response Body Format	Scenario
200	DX-AS-TIP-Success-Response	Successful introspection.
400	DX-AS-Error-Response	Missing information, invalid input
401	DX-AS-Error-Response	Missing authentication token, invalid authentication token etc.
403	DX-AS-Error-Response	Resource server not allowed to inspect the presented token

8 COMMON BEHAVIOURS

8.1 Common API Response Template

This section defines the common response templates/ schemas for all the DX Interface API response payloads. The response template shall follow the structure as recommended in IETF RFC7807. In particular, the framework to provide additional finer details about the errors in a machine-interpretable way is adopted. The response payloads contain the following mandatory attributes:

a) **type**: A URN (Uniform Resource Name), as defined in IETF RFC 8141, describing the exact "type" of the response. This allows service implementations to add additional machine-interpretable codes to specify finer-grained details about the errors/issues apart from the high-level status codes presented in the header of the response. A recommended list of above types in URN format can be found in **Annex C** (informative).

b) **title**: Human readable title of the response.

Depending upon the API response the payload may contain the following optional attributes:

a) **detail**: Detailed human readable explanation in case of errors.

b) **results**: Array of returned objects as per the result of the operation.

c) **totalHits**: Total number of records found for the search query.

d) **limit**: Applicable in case of pagination response (*See* Section **8.4**).

The contents of **results** depend upon the requested API. For example, it may contain data for a search query on a given resource or it may contain subscription information or data for DX catalogue search etc. The information about the returned objects within **results** attributes should be documented along with the individual API descriptions or should be documented

via additional means such as API objects, DX entity schemas, Data descriptors for given resources etc. Similarly, a detailed list of URN based type codes for each DX service should be made available to the consumers.

8.2 JSON Payloads

The HTTP APIs in this standard have been defined for JSON payloads. JSON, as defined in IETF RFC 8259, is a well accepted and preferable format for exchanging data for HTTP APIs.

The APIs accept JSON documents for request payload and also send responses as JSON documents. The response header for all the APIs returning data shall respond with parameter 'Content-Type' set to 'application/json'. It is recommended that for the current version of API the accept header parameter allowed by the servers may be 'application/json' or 'application/json+ld'.

8.3 API Security

8.3.1 HTTP over TLS

This standard mandates that all the APIs shall use TLS protection over its API endpoints. All APIs specified in this standard shall support HTTP over TLS, as specified in IETF RFC 2818 (also *see* IETF RFC 5246).

8.3.2 Input Validation

API implementations shall validate parameters passed as query, header or request body parameters. It is recommended that the validation strategies should follow API security best practices, such as recommendations by OWASP API Security Project, to provide protection against SQL injection, provide range and limit validations etc. Additionally, checks shall be performed to not accept request body with disallowed characters as specified in Section 4.6.4 in the NGSI-LD API Specifications.

8.3.3 Request/Response Payload Size Limitations

API implementations shall specify a maximum size limit (e.g., in MB) on the response/request payloads. Implementations shall also specify a maximum limit on the number of records to be returned in response to successful search queries, where the records are returned in a result array. API server implementations supporting query paginations shall additionally specify a limit on the maximum value for the parameter **offset** (*see* Section **8.4**). API Requests leading to violations of the above limits shall return an error code of 400. Additional information about the type of error code should be provided using **type** parameter in the response payload (*see* Section **8.1**).

8.3.4 General Recommendations

In general, it is recommended to incorporate all the API security best practices, such as recommended by OWASP API Security Project, NIST SP 800-204 etc., while implementing the interfaces proposed in this standard. Further, it is recommended to incorporate API gateways, providing further protection from attacks such as DDOS attacks, as an integral part of the implementation architectures.

8.4 Query Pagination

A DX API can potentially return a result set including a large number of records. This may typically apply for search queries supported by both the Catalogue service and the Resource Access Service. For such scenarios, API implementations shall support pagination of query results. The response pagination behaviour is controlled by the following query parameters:

a) **limit** (Integer): Specifies the number of records to be responded back in a given pagination iteration. This is an optional parameter and in case it is not provided a default maximum limit as per the implementation shall apply.

b) **offset** (Integer): Specifies an offset value specifying the record number from which the response limit shall be applied for a given pagination iteration.

An implementation shall specify a default limit on the maximum number of records that can be returned in a single pagination iteration and parameter **limit** with values higher than this shall not be accepted. Further, an implementation shall also specify a limit on the maximum acceptable value for parameter offset. Implementations shall respond back with an error code 400 in case input parameters limit and offset exceed the acceptable values.

The parameter **limit** contained in the pagination iteration response specifies the actual number of records returned in each iteration. In case fewer than requested records are returned this parameter will be useful to compute the offset for the next page. The pagination termination ends when the computed offset for the next page is greater than equal to **totalHits** or greater than the maximum allowed offset.

Implementations shall also ensure that response for each pagination iteration shall not exceed the maximum payload size limit. Server implementations may respond back with an error code 400. Or it may respond back with a reduced number of records.

Server implementations may support additional advanced pagination mechanisms, for example, time-based pagination, cursor-based pagination and/

or offset based pagination as specified in https://developers.facebook.com/docs/graph-api/using-graph-api/#paging. Any such paging mechanism implementations shall adhere to the general pagination principles specified in Section 4.12 of NGSI-LD API Specifications.

8.5 Additional Common Status Codes

Apart from the functional status codes specified in API endpoint descriptions, additional status codes, e.g., 500 (Internal Server error), 408 (Request Timed out) etc., may be returned by API server implementations. It is recommended that the server implementations should adhere to the well-established norms of categorisation of status codes based on the first digit of the code, i.e., 1xx (Informational), 2xx (Successful), 3xx (Redirection), 4xx (Client Error), 5xx (Server Error), as specified in IETF RFC 7231. It is further recommended that additional URN-based codes be defined (e.g., as described in Annex C) to provide an additional description of the type of response associated with a given status code.

8.6 Base URL

The base URL precedes all API endpoints and verbs. A template for base-url is as follows: https://{Root_Path}/{version}/. Table 80 describes the base URL components.

Table 80 Base URL Components

Name	Description
Root_Path	A root path contains the server address, port, protocol and implementation specific path names (if any) with which the APIs are exposed.
version	Specifies the API version associated with the endpoint. Note that for the APIs specified in this document the 'version' should be set to **v1**.

It is recommended to include DX service identity in the {Root_path}. The recommended values for DX service ids are as follows:

a) DX Catalogue Service - **cat**

b) DX Authorization Service - **auth**

c) DX Resource Access Service - **ngsi-ld** or **rs**

NOTE - having a version number in the API will help the server to support multiple versions of the same endpoint. With this approach it will be easier for the APIs to be constantly updated as and when there are new requirements.

Using the above base-url template, an example for the full path for an endpoint, e.g., /search endpoint, of DX Catalogue Interface can be: https://<domain-ip>/dx/cat/v1/search, where

a) Base-url is https://<domain-ip>/dx/cat/v1

1) Root Path is *https://<domain-ip>/dx/cat* (Note that service id is cat)

2) Version is *v1*

b) Endpoint is */search*

Similarly, examples of AS tokens endpoint and RS entities endpoint are: https://auth.dx.org.in/dx/auth/v1/tokens and https://rs.dx.org.in/dx/ngsi-ld/v1/entities, respectively.

8.7 API Usage Logs

Implementation of all APIs shall log user interactions and make available the usage statistics for auditing purposes. It is recommended that all the best practices for logging for auditing purposes should be followed.

8.8 API Documentation

Service implementations of DX interfaces shall make available API documentation that shall also include all implementation specific information such as default parameter values, implementation specific limits, optional response payloads, additional catalogue information model entities etc. It is recommended that the documentation be provided in language agnostic format easy to be read by both humans and machines such as OpenAPI Specification format.

ANNEX A

(*Clause* 5.1.6)

EXAMPLE CATALOGUE ITEMS

A-1 EXAMPLES OF CATALOGUE ITEMS IN JSON FORMAT

Some examples of catalogue items of various types listed in **5.1** are given below.

Example of Resource Item

```
{
    “id”: “<provider>/<resourceServer>/<resourceGroup>/<resource>”,
    “type”: [“iudx:Resource”, “iudx:EnvAQM”],
    “name”: “myResource1”,
    “resourceGroup”: “<provider>/<resourceServer>/<resourceGroup>”,
    “provider”: “<provider>”,
    “tags”: [“tags”, “describing”, “resource”],
    “description”: “This is a description of myResource1 which is a co2 sensor”,
    “location”: {
        “type”: “Place”, “address”: “Bangalore”, “geometry”: {
          “coordinates”: [ 77.57003, 13.01417 ], “type”: “Point” }
      }
}
```

Example of Resource Group item

```
{
    “id”: “<provider>/<resourceServer>/<resourceGroup>”,
    “type”: [“iudx:ResourceGroup”, “iudx:EnvAQM”],
    “name”: “<resourceGroup>”,
    “provider”: “<provider>”,
    “tags”: [“tags”, “describing”, “resource”, “group”],
    “description”: “This is a description of <resourceGroup> of co2 sensors “,
    “resourceType”: “MESSAGESTREAM”,
    “iudxResourceAPIs”: [“TEMPORAL”, “SPATIAL”, “ATTR”],
    “accessPolicy”: “SECURE”,
    “authControlLevel”: “GROUP”,
    “dataDescriptor”: {
        “co2”: {
          “type”: [ “TimeSeriesAggregation” ],
          “description”: “The aggregated value for Carbon Dioxide (C02) in part
per million over the last 15 minutes.”,
          “avgOverTime”: {
            “dataSchema”: “iudx:Number”,
            “unitText”: “part per million (ppm)”,
            “unitCode”: “59”,
            “type”: [
              “ValueDescriptor”
            ],
            “aggregationDuration”: {
              “unitCode”: “MIN”,
              “unitText”: “minutes”,
              “value”: 15
            },
            “description”: “Average value of CO2 for the last 15 minutes”
          }
        }
    }
}
```

Example of a Data Descriptor entity


```
 {
       “co2”: {
         “type”: [ “TimeSeriesAggregation” ],
         “description”: “The aggregated value for Carbon Dioxide (C02) in part
per million over the last 15 minutes.”,
         “avgOverTime”: {
           “dataSchema”: “iudx:Number”,
           “unitText”: “part per million (ppm)”,
           “unitCode”: “59”,
           “type”: [
             “ValueDescriptor”
           ],
           “aggregationDuration”: {
             “unitCode”: “MIN”,
             “unitText”: “minutes”,
             “value”: 15
           },
           “description”: “Average value of CO2 for the last 15 minutes”
         }
       }
   }
```


Example of Provider item


```
{
    “id”: “<provider”,
    “name”: “<provider-name>”,
    “description”: “A description of this provider”,
    “providerOrg”: {
        “location”: {
            “type”: “Place”, “address”: “Bangalore”,

             “geometry”: {
                 “coordinates”: [ 77.57003, 13.01417 ],

                “type”: “Point” }
                },
           “additionalInfoURL”: “https://dx.org.in”,
           “name”: “dx”
    },
    “type”: [
        “iudx:Provider”
    ]
}
```

Example of ResourceServer item

```
{
	“type”: [
	“iudx:ResourceServer”
	],
	“id”: “<resource-server>”,
	“name”: “rs.dx.org.in”,
	“description”: “DX resource server”,
	“tags”: [ “DX”, “Resource”, “Server”, “Platform” ],
	“resourceServerHTTPAccessURL”: “rs.dx.org.in”,
	“resourceServerStreamingAccessURL”: “databroker.dx.org.in”,
	“resourceServerOrg”: {
	“name”: “dx”,
	“additionalInfoURL”: “https://dx.org.in”,
	“location”: { “type”: “Place”, “address”: “Bangalore”,
		“geometry”: { “type”: “Point”, “coordinates”: [ 77.570423,
13.013945 ]
		}
	}
	},
	“location”: {
	“type”: “Place”, “address”: “Bangalore”,
	“geometry”: { “type”: “Point”, “coordinates”: [ 77.570423, 13.013945 ] }
	}
}
```

A-2 REPRESENTATION OF CATALOGUE ITEMS USING JSON-LD FORMAT

This section is non-normative in nature.

A given DX catalogue implementation may choose advanced JSON-based representations for the meta-information objects. This section discusses representing catalogue entities using JSON-LD as described in W3C JSON-LD 1.1 specifications. Note that JSON-LD document is a valid JSON document.

The catalogue items can be represented as *linked data* objects (*see* Linked Data). The mandatory attributes, and preferably the custom attributes as well, of an item, are mapped to discoverable universal resource identifiers, as defined in IETF RFC 3986. That is, the attributes are provided with a *context*. The context for a given attribute may contain information on how the attribute should be interpreted. The context enables providing linkages, using linked data primitives, to vocabularies/taxonomies thereby leading to further enhancement in its interpretability by machines (and humans).

JSON-LD framework may be used to provide linked data encodings to the attributes in the catalogue objects. All the catalogue items are JSON-LD documents and necessarily need to include “@context” field, which contains JSON-LD context, that maps attributes to IRIs (Internationalized Resource Identifiers) as described in IETF RFC 3987 providing unambiguous identification of these attributes.

For attributes in the catalogue items, the context can be provided via a hosted DX vocabulary. For custom attributes, it is recommended that the provider of these items should use context from DX vocabulary and/or from other existing vocabularies.

JSON-LD ‘@type’ field is used to define the type of a catalogue item. Note that a catalogue item is allowed to have multiple types and it simply implies that the item may contain attributes from multiple types. One such scenario arises when a given ‘resource’ item needs to contain a domain specific meta information. For example, an air quality monitoring data resource may mention the ‘model of the device’ which is a domain specific data model attribute. In such a case, the catalogue item will include an ‘AQM data model’ type along with ‘Resource’ item type. Another such scenario arises where a ‘DataDescriptor’ object, which typically belongs to a ‘Data Model’ type, may contain attributes from multiple ‘Data Model’ types.

As mentioned above, an item type for a catalogue item has an associated schema which is used to define the syntactic structure of an item belonging to this type.

These schemas are specified as a part of catalogue entity definitions in DX specific vocabulary. An equivalent representation may also be specified using JSON schema framework. In particular, JSON-schema based schemas can also be used for validating the structure of catalogue items.

In the table below, an example representation of a data descriptor object using JSON-LD is presented. Note that, in the context object, a vocabulary (IUDX vocabulary with URL https://voc.iudx.org.in/) has been set as the default vocabulary. Additional vocabularies can be referred to as and when required. *See* for example, "qudt" vocabulary (https://qudt.org/vocab/unit) has been used for choosing units for the quantity 'atmosphericPressure'. Similarly, for some other established attributes, such as unitCode and unitText, a popular vocabulary schema.org (https://schema.org/) has been used. JSON-LD parsers, such as one available in JSON-LD Playground[2)], are able to expand such JSON-LD objects and provide all the necessary linkages to the applications consuming this data.

Example of DataDescriptor object using JSON-LD

```
{
    "@context": [
    "https://voc.iudx.org.in/",
     {
      "qudt-unit": "http://qudt.org/vocab/unit/",
      "unitCode": {
        "@type": "@id"
      },
      "schema": "https://schema.org/",
      "iudx": "https://voc.iudx.org.in/"
    }],
    "type": [ "iudx:EnvAQM", "iudx:DataDescriptor"],
    "atmosphericPressure": {
       "type": [ "ValueDescriptor" ],
       "description": "Measured Air pressure",
       "schema:unitCode": "qudt-unit:MILLIBAR",
       "schema:unitText": "Milli Bar",
       "dataSchema": "iudx:Number"
    }
}
```

2) JSON LD Playground: https://json-ld.org/playground/

ANNEX B

(*Clause* 6.1.2.8)

RESOURCE SERVER DATA INGESTION

This Annex describes two optional APIs that may be implemented by the resource access service for data ingestion.

B-1 API ENDPOINTS FOR INGESTION

B-1.1 Endpoint: /entities

The /entities API allows DX providers to publish data into the DX platform using a POST method. The data published into DX shall be as per the data descriptor defined in the DX Catalogue.

The POST method of the /entities API shall be a protected endpoint. To publish data using this API, a valid authorization token is mandatory in the header parameters of the request.

The request body for this API shall contain the data to be published in JSON format. Note that the data payload shall always contain the resource "id", as per the DX Catalogue, apart from the mandatory data attribute/value object pairs. As a best practice, it is recommended that the publish service, accessible through this endpoint, in DX resource server validate the published data with the data descriptor present in the DX Catalogue. Table 81 summarizes the applicable parameters for publish functionality.

Table 81 Parameters and Status codes for Publish Data

Functionality	Publish Data into DX
Methods	POST
Required header parameters	token
Required body parameters	id, Mandatory attributes and value as per the data descriptor in the DX catalogue
Status codes	201, 400, 401, 404

B-1.1.1 *API Response*

Table 82 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

B-1.2 API Endpoint: /ingestion

The /ingestion API allows DX providers to register, list and delete a resource ingestion stream for one or more data resources through a streaming service, such as AMQP , MQTT etc. The specific streaming service supported is implementation dependent and is out of scope of the current specifications.

Table 82 Response codes for POST method for Resource Access Service API endpoint /entities

HTTP Status code	Response Body Format	Scenario
201	DX-RS-CreatedSuccess-Response	Publication successful
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Token, Invalid Token, Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX

The ingestion API shall be a protected API. A valid authorization token is mandatory in the header parameters of the request. Information about the resource id of the resource and security scope of the resource can be obtained from the associated meta-information in the DX catalogue.

For future extensibility, to allow for ingestion modes other than streaming, such as remote API call integration, this API shall require a header parameter **options**. For the current specifications, which only support streaming mode of ingestion, this parameter should always be set equal to 'streaming'.

The request body schema for this API is defined by entity DX-RS-ReqBody-Ingestion described in **6.1.5**. A DX provider can use the POST method to register a resource for ingestion.

Tables 83-84 summarize the applicable parameters for ingestion functionality.

Table 83 Parameters and Status codes for ingestion registration

Functionality	Register a set of resources for ingestion
Methods	POST
Required body parameters	entities
Status codes	201, 400, 401, 404, 409

Table 84 Parameters and Status codes for listing and deleting resources in ingestion

Functionality	List/Delete resource(s) registered for ingestion
Methods	GET, DELETE
Required path parameters	ingestionID
Status codes	200, 400, 401, 404

B-1.2.1 *API parameters*

DX resource ingestion APIs accept the following request body parameters as described in section **B-2.1**: entities

In addition to the above the parameters given in Table 85 are required:

Table 85 API parameters for Resource Access Service API endpoint /ingestion

Parameter Name	Parameter Value Data Type	Parameter Type	Description
ingestionID	String	Path	Represents the unique ingestionID for a resource ingestion (*See* DX-RS-Ingestion-Params object below for more details.)
options	String	Header	Represents the streaming options requested by the consumer.

B-1.2.2 *API Response*

Table 86 describes the status codes and response body formats. The response body formats are explained in **6.1.5**.

Table 86 Response codes for Resource Access Service API endpoint /ingestion

HTTP Status code	Response Body Format	Scenario
201	DX-RS-CreatedSuccess-Response	Resource Ingestion registration successful
200	DX-RS-DeletedSuccess-Response	Registered resource ingestion deleted successful
400	DX-RS-Error-Response	Invalid Syntax, Invalid param type, Invalid param value etc.
401	DX-RS-Error-Response	Missing Token, Invalid Token, Token expired etc.
404	DX-RS-Error-Response	Resource not present in DX

B-1.3 Input validation

It is recommended that the DX Resource Access Interface should allow data to be published only as per the data structure and format, specified by the data descriptor, defined in the DX Catalogue. Further, it is recommended that the resource server ensures additional input validation as defined in section 4.6.4 in the NGSI-LD API Specifications.

B-2 OBJECT DEFINITIONS FOR INGESTION APIS

B-2.1 DX-RS-ReqBody-Ingestion

Table 87 describes the request payload schema for an ingestion registration (/ingestion) API.

Table 87 Request payload schema for Resource Access Service API endpoint /ingestion

Attribute	Data Type	Attribute Value
entities	Array[String]	Array of resource id(s) as per DX catalogue.

B-2.2 DX-RS-Ingestion-Params

Table 88 Request payload schema for update operation for endpoint /ingestion

Attribute	Data Type	Attribute Value
ingestionID	String	Represents the unique ingestionID for a resource ingestion.
entities	Array[String]	Array of resource id(s) associated with ingestion registration.

Resource Access Service should ensure that ingestionID is unique for all ingestion streams issued by it. The format of ingestionID is left out of scope of this standard and is dependent on a given implementation.

The above object may contain additional ingestion parameters, such as ingestion credentials, streaming server URLs etc., which are dependent on the specific streaming service implemented by the resource access service. Details of these specific parameters shall be made available via implementation specific API documentation for a DX resource server.

ANNEX C

(*Clause* 8.1, *informative*)

DX Response Codes

The response code of HTTP has its own limitation on the extent to which an error associated with the query can be communicated. In order to avoid the above issue and in order to allow application developers to build the application with a self correcting feature, DX recommends the interfaces to provide the following URNs which are represented in a machine-interpretable format.

The URN formats are as per the recommendations in the IETF RFC 2141 and have the following syntax:

a) urn
b) Namespace Identifier (NID)
c) Namespace Specific String (NSS)

The format of the urn shall be urn:<NID>:<NSS>

The leading "urn:" sequence is case-insensitive. It is recommended to use "dx" as the NID and "<interfaceName>":"<statusCode>", e.g. "cat:Success" as NSS.

The recommended URN's for the DX interfaces are detailed in section C-1, C-2, C-3.

C-1 CATALOGUE SERVICE RESPONSE URN

Table 89 specifies the recommended API response URNs for a DX catalogue interface.

Table 89 Catalogue Service API Response URNs

URN	Description
urn:dx:cat:Success	Successful operation
urn:dx:cat:WrongProvider	Operation not permitted for given provider link
urn:dx:cat:WrongResourceServer	Operation not permitted for given resource server link
urn:dx:cat:WrongResourceGroup	Operation not permitted for given resource group link
urn:dx:cat:InvalidSchema	Invalid schema for the given document type
urn:dx:cat:NonExistentID	No record of the given id
urn:dx:cat:AlreadyExists	Document already exists
urn:dx:cat:ExpiredAuthorizationToken	Authorization token has expired
urn:dx:cat:MissingAuthorizationToken	Token needed and not presented
urn:dx:cat:InvalidAuthorizationToken	Token is invalid
urn:dx:cat:ItemNotFound	Document of given id does not exists
urn:dx:cat:InvalidGeoParam	Invalid geo param for the given query
urn:dx:cat:InvalidGeoValue	Invalid geo param value for the given query
urn:dx:cat:InvalidProperty	Invalid property for the given query
urn:dx:cat:InvalidPropertyValue	Invalid property value for the given query
urn:dx:cat:BadTextQuery	Bad text value for the query
urn:dx:cat:MethodNotAllowed	Method not allowed for given endpoint
urn:dx:cat:UnsupportedMediaType	Requested/Presented media type not supported
urn:dx:cat:InvalidRelationshipType	Invalid relation type for the given query
urn:dx:cat:InvalidRelationParent	Parent document for the given query doesn't exists
urn:dx:cat:InvalidListType	Invalid list type for the given query
urn:dx:cat:responsePayloadLimitExceeded	Search operations exceeds the default response payload limit
urn:dx:cat:requestPayloadLimitExceeded	Operation exceeds the default request payload limit
urn:dx:cat:requestOffsetLimitExceeded	Operation exceeds the default value of offset
urn:dx:cat:requestLimitExceeded	Operation exceeds the default value of limit

C-2 AUTHORIZATION SERVICE RESPONSE URN

Table 90 specifies the recommended API response URNs for a DX authorization interface.

C-3 RESOURCE ACCESS SERVICE RESPONSE URN

Table 91 specifies the recommended URNs for a DX resource access interface.

Table 90 Authorization Service API Response URNs

URN	Description
urn:dx:as:Success	Successful operation
urn:dx:as:MissingInformation	Necessary parameters missing
urn:dx:as:InvalidInput	Invalid request param / value
urn:dx:as:InvalidRole	User does not have the required role to call the API
urn:dx:as:AlreadyExists	If the AS entity to be added or created already exists
urn:dx:as:MissingAuthenticationToken	Authentication token required
urn:dx:as:InvalidAuthenticationToken	Authentication token is invalid or expired

Table 91 Resource Access Service API Response URNs

URN	Description
urn:dx:rs:success	Successful operation
urn:dx:rs:InvalidTemporalParam	Invalid temporal param for the given query
urn:dx:rs:InvalidTemporalRelationValue	Invalid temporal param value for the given query
urn:dx:rs:InvalidTemporalDateFormat	Invalid temporal param value date format for the given query
urn:dx:rs:InvalidGeoParam	Invalid geo param for the given query
urn:dx:rs:InvalidGeoValue	Invalid geo param value for the given query
urn:dx:rs:InvalidAttributeParam	Invalid attribute param for the given query
urn:dx:rs:InvalidAttributeValue	Invalid attribute param value for the given query
urn:dx:rs:InvalidOperation	Operation requested in the endpoint is not permitted
urn:dx:rs:UnauthorizedEndPoint	Access to the endpoint is not available
urn:dx:rs:UnauthorizedResource	Access to the resource is not available
urn:dx:rs:ExpiredAuthorizationToken	Token has expired
urn:dx:rs:MissingAuthorizationToken	Token needed and not presented
urn:dx:rs:InvalidAuthorizationToken	Token is invalid
urn:dx:rs:ResourceNotFound	Document of given id does not exists
urn:dx:rs:MethodNotAllowed	Method not allowed for given endpoint
urn:dx:rs:UnsupportedMediaType	Requested/Presented media type not supported
urn:dx:rs:responsePayloadLimitExceeded	Search operations exceeds the default response payload limit
urn:dx:rs:requestPayloadLimitExceeded	Operation exceeds the default request payload limit
urn:dx:rs:requestOffsetLimitExceeded	Operation exceeds the default value of offset
urn:dx:rs:requestLimitExceeded	Operation exceeds the default value of limit

ANNEX D

(*Clause* 4)

DX Usage Examples and Interaction Flows

This annex presents two illustrative interaction flows that describe interactions between DX interfaces and DX clients. A consumer flow describes various steps involved in a DX consumer getting access to data from an access controlled data resource. A provider flow describes various steps involved in onboarding a resource on DX catalogue and setting up of a resource access policy for that data resource.

D-1 CONSUMER FLOW

This section explains the interaction flow for a DX consumer who is interested in accessing data from a given resource. For the purpose of describing this flow, the following assumptions have been made: (a) DX consumer has already registered with the data exchange using AS endpoint '/user/profile' (*see* Section **7.1.3.1**) with a 'consumer' role. (b) Provider of the resource of interest has already consented to provide access to the above consumer and has already set policy to allow data access for the above consumer.

Fig. 3 describes the consumer flow interaction diagram. A brief explanation is as follows:

D-1.1 Discover Resources

In Step [1] in Fig. 3, the DX consumer uses the Catalogue Server search endpoint '/search' to discover

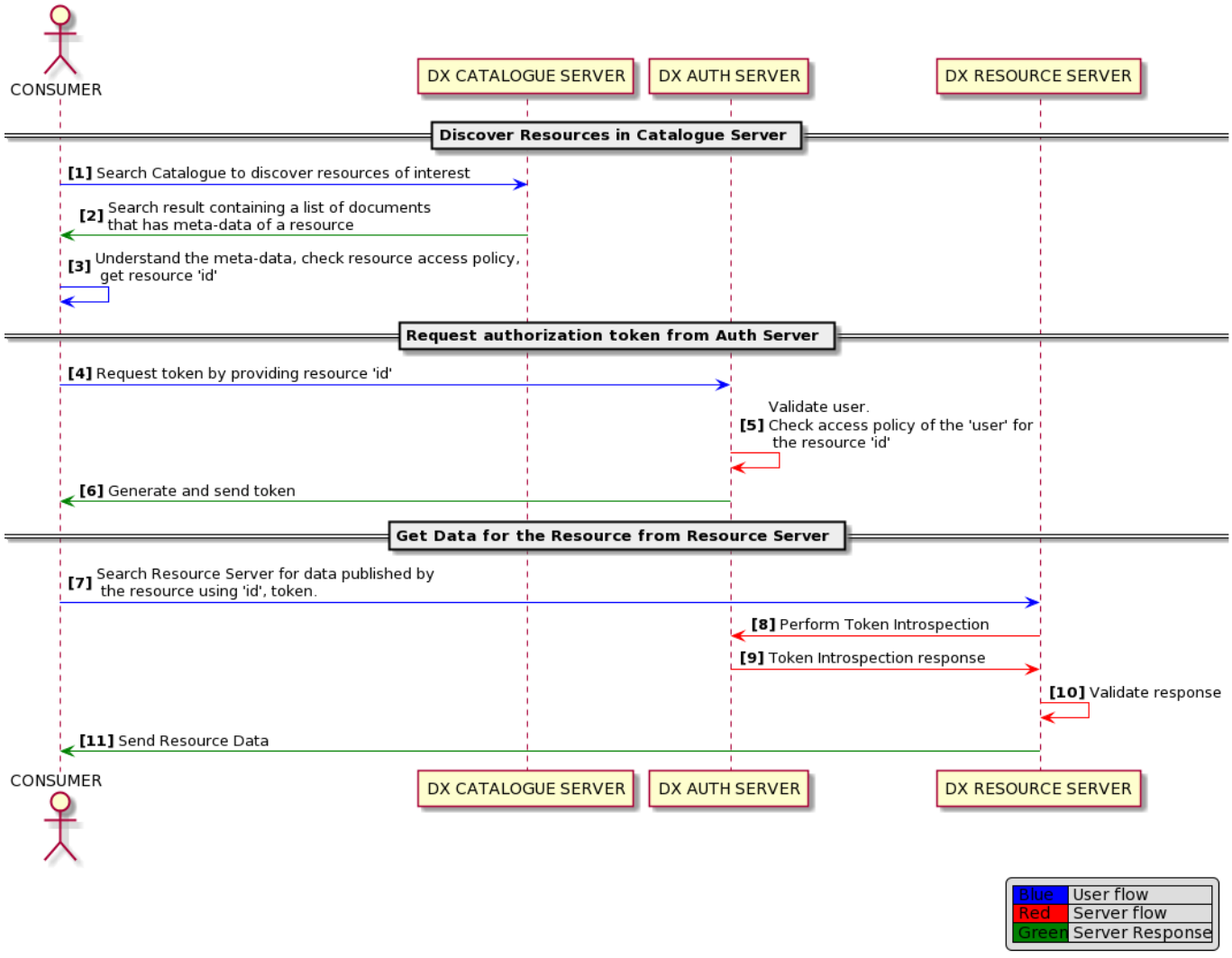

FIG. 3 CONSUMER FLOW INTERACTION DIAGRAM.

resources of interest. For all the search options and query semantics *see* **5.2**. As an example, let us consider that a consumer is looking to identify "air quality" data resources. The text search feature may be used in this case. The cURL example to search for resources providing "air quality" data in DX Catalogue Server is as shown below.

```
curl --location -g --request GET 'https://catalogue.dx.org/iudx/cat/v1/search?q=air%20quality&property=[type]&value=[[Resource]]'
```

An example successful response is as follows where catalogue server returns back with resource items corresponding to the matching criteria:

```
{
"type":"urn:dx:cat:Success ",
"title":"Successful Search Query",
"results":[
   {
        "id": "resource-id-xyz-abc",
        "description": "Air quality monitoring devices/sensors in XYZ city." ,
        "itemStatus": "ACTIVE",
        "provider": "urn:cat:4a15c9960ffda227e9546f3f46e629e1fe4132b",
        "resourceServer": "urn:cat:27e503da0bdda6efae3a52b3ef423c1f9005657a",
        …..
        …..
        "resourceType": "MESSAGESTREAM",
    }, …….
 ],
  "totalHits": 108
}
```

The CS response contains resource entities that help consumers get different kinds of meta-information associated with the resources. Most importantly it provides consumer applications with the 'id' of the resource which will be used in subsequent interactions, namely authorization token request and data access request.

D-1.2 Request Authorization token in the Auth Server

In Step [4] in Fig. 3, the DX consumer uses AS API endpoint '/tokens' (*see* Section **7.1.3.3**) to get an authorization token to access the resource in the resource server. As an example, let us consider that the consumer is interested in getting a token for the resource with id 'resource-id-xyz-abc'. The cURL example to get a token from the DX-Auth-Server for accessing the resource is as shown below.

```
curl --location --request POST 'https://authorization.dx.org/auth/v1/token'
--header 'Content-Type: application/json'
--header 'Authorization: Basic OTY2NTNmOGUtODBiZS0xMWU2LWIzMmItYzdiY2RlODY2MTNhOkUtaFJlVk11UnlaYnlyMUdpa2llRXc0SnNsYU02c0RwYjE4XzlWNTlQRnc='
--data-raw '{
   "request": [
      "resource-id-xyz-abc"
   ]
}'
```

As mentioned above, the policy has already been written to provide access to this consumer by the provider of this resource. Hence, Step [5] in Fig. 3 will result in a successful response. An example of a successful response where the AS responds back with the authorization token is as shown below:


```
{
“type”: “DX-AS-Token-Success-Resp”,
“title”: “Token generation successful”,
“results”:[
{
  “token”: “eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c”,
  “expiry”: “2021-03-10T12:33:07.740Z”
}]
}
```


Note that the authorization tokens have an associated expiry time. Once expired the authorization token needs to be refreshed using the same endpoint as above. The token from the above response (Step [6]) should be used by the consumer to access data for this resource.

D-1.3 Get data for the resource in the Resource Server

In Step [7] in Fig. 3, the DX consumer uses the RS endpoint ‘/entities’ to access the data. For all the search options and query semantics *see* **6.1**. For example, let us consider that the consumer is interested in obtaining the latest data published by a resource using the /entities API (*see* **6.1.3.1**). The cURL example to get the latest data of a resource, with id ‘resource-id-xyz-abc’, from the DX-Resource-Server is as shown below:

```
curl --location --request GET ‘https://rs.dx.org/ngsi-ld/v1/entities/resource-id-xyz-abc’
--header ‘token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c’
```

The DX Resource Server on receiving this request uses the DX Auth Server endpoint ‘/tokens/introspect’ to introspect the token (Step [8]). If the token provided by the DX Auth Server is a JWT token, then the DX Resource Server can decode it by itself. Else it shall use the TIP endpoint: ‘/tokens/introspect’ (*see* **7.1.3.4**) to decode the token. Based on the TIP response, DX Resource Server can validate the request (Step [10] in Fig. 3) as coming from an authorized consumer and if validation succeeds it then responds back with the data. An example successful response (Step [11] in Fig. 3) is as shown below:


```
{
“type”: “DX-RS-Success-Response-Search”,
“title”: “Search successful”,
“results”:
[
  {
   “id”: “resource-id-xyz-abc”,
    “ambientNoise”: {
      “avgOverTime”: 81.06
    },
```

```
    “observationDateTime”: “2021-06-04T17:01:01+05:30”,
    “pm2p5”: {
      “avgOverTime”: 0.29
    },
    “co”: {
      “avgOverTime”: 0.5
    },
    “deviceStatus”: “ACTIVE”,
    “atmosphericPressure”: {
      “avgOverTime”: 0.93
    },
    “airQualityLevel”: “SATISFACTORY”
  }
],
“totalHits” : 1
}
```


D-2 PROVIDER FLOW

This section explains the interaction flow for a DX provider who is interested in onboarding a data resource and also providing access control for that resource. For the purpose of describing this flow, the following assumptions are made:

a) DX Provider has already registered with the data exchange using AS endpoint ‘/user/profile’ (*see* **7.1.3.1**) with a ‘provider’ role.

b) DX Provider has obtained consent from DX admin to access DX catalogue server during resource registration and DX resource server during data publishing.

Fig. 4 describes the provider flow interaction diagram and **D-2.1** explain various steps listed in Fig. 4.

D-2.1 Register Resources in Catalogue Server

In Step [4] in Fig. 4, the DX provider uses the Catalogue endpoint ‘/item’ (*see* **5.2.2.1**), to create a resource in DX Catalogue Server. DX Catalogue endpoint is accessed using a token obtained from DX Auth Server using Step [1] to Step [3] which is similar to the consumer flow described earlier (*see* section Request Authorization token in the Auth Server).

The cURL example to register resources in DX Catalogue Server is as shown below.

```
curl --location --request POST ‘https://catalogue.dx.org/iudx/cat/v1/item’
--header ‘Content-Type: application/json’
--header ‘token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c’
--data-raw ‘{
  “@context”: “context-url”,
  “name”: “some-name”,
  “type”: [ “Resource”],
  “instance”: “name of the instance to which this item belongs”
}’
```

On successful creation of an item the provider will get the following response:

```
{
 “type”: “urn:dx:cat:Success”,
 “title”:”Resource registered successfully”,
 “results”: [
   {
    “id”: “resource-id-xyz-abc”,
    “method”: “insert”
   }
 ]
}
```

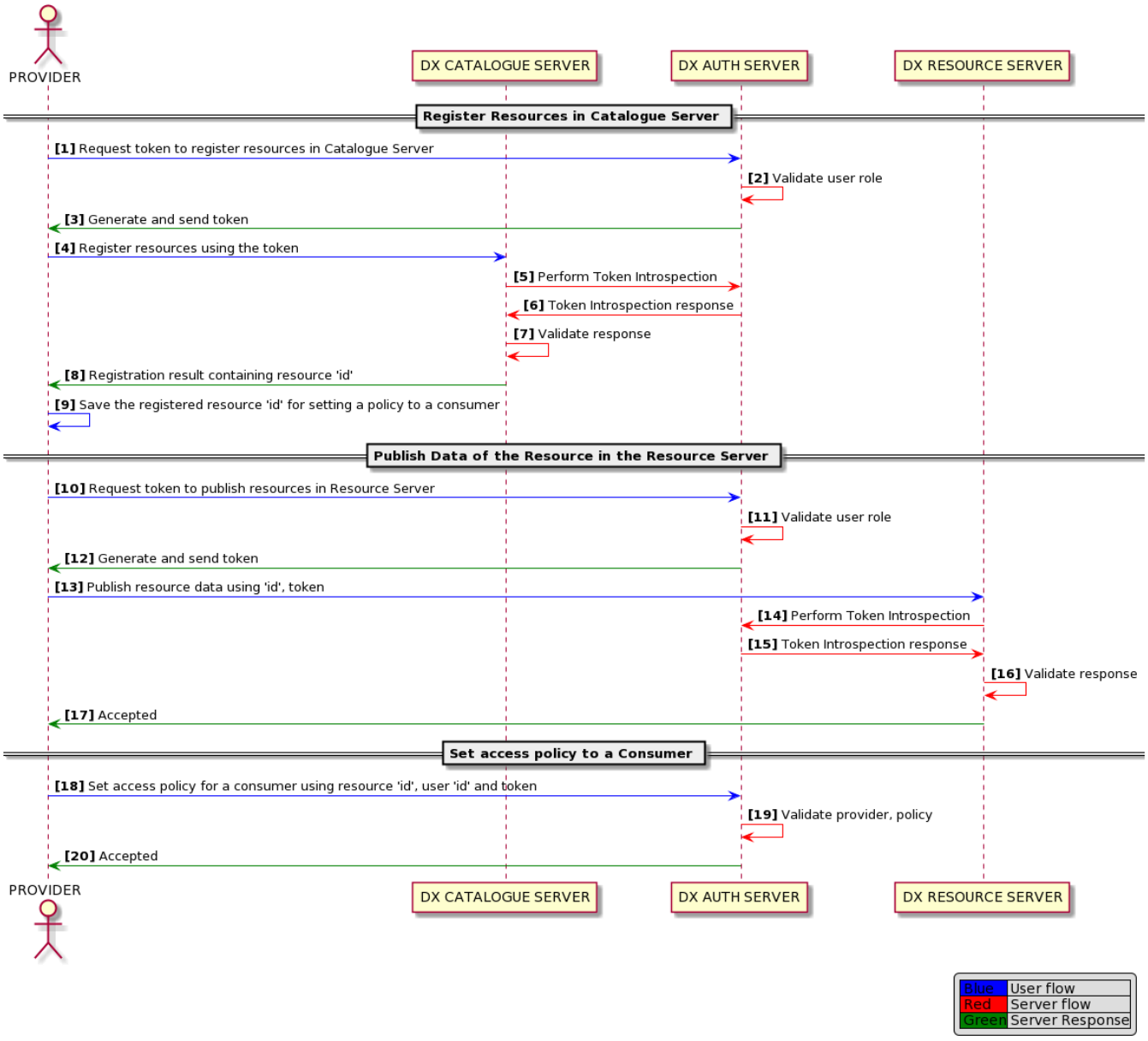

FIG. 4: PROVIDER INTERACTION FLOW DIAGRAM

On successful registration, as shown in Step [8] in Fig. 4, the DX Catalogue Server shall respond with the DX resource 'id' which can be used by the DX Provider to set policies to a DX Consumer. Note that any subsequent modifications to this catalogue item, i.e., with id "resource-id-xyz-abc", can only be performed by its provider.

D-2.2 Publish data of the Resource in the Resource Server

It is to be noted that publishing data of a resource as explained in Step [10] to Step [17] in Fig. 4 is applicable only for providers wanting to push data to an external resource server to host its data resource.

The following example assumes that the DX Provider is using an external DX Resource Server with the '/entities' endpoint as described in section **B-1** (API Endpoints for Ingestion).

After successfully obtaining the token from the DX Auth Server as per Step [10] to Step [12] in Fig. 4, the DX Provider can use the API as shown in the below cURL example to publish data into the DX Resource Server.

```
curl --location --request POST 'https://rs.dx.org/ngsi-ld/v1/entities'
--header 'token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c'
--header 'Content-Type: application/json'
--data-raw ' {
  "id": "resource-id-xyz-abc",
   "ambientNoise": {
     "avgOverTime": 81.06
   },
   "observationDateTime": "2021-06-04T17:01:01+05:30",
   "pm2p5": {
     "avgOverTime": 0.29
   },
   "co": {
     "avgOverTime": 0.5
   },
   "deviceStatus": "ACTIVE",
   "atmosphericPressure": {
     "avgOverTime": 0.93
   },
   "airQualityLevel": "SATISFACTORY"
 }'
```

An example successful response to a publish operation is as described below:

```
{
 "type": "urn:dx:rs:Success",
 "title":"Resource published successfully",
 "results": [
   {
     "id": "resource-id-xyz-abc"
   }
 ]
}
```

Optionally, while connecting to an external DX Resource Server, the DX Provider can also use the streaming / ingestion API as defined in endpoint: '/ingestion' (*see* **B-1.2**) in case it is available in the DX Resource Server.

D-2.3 Set access policy to a Consumer

In Step [18] in Fig. 4, the DX provider can use the Auth Server endpoint '/policies' (*see* Section **7.1.3.2**), to set access policy for a consumer.

The cURL example to set access policy in DX Auth Server is as shown below.

```
curl --location --request POST 'https://authorization.dx.org/auth/v1/policies'
--header 'Authorization: Basic
OTY2NTNmOGUtODBiZS0xMWU2LWIzMmItYzdiY2RlODY2MTNhOkUtaFJlVk11
UnlaYnlyMUdpa2llRXc0SnNsYU02c0RwYjE4XzlWNTlQRnc='
--header 'Content-Type: application/json'
--data-raw '{
    "item_id":"resource-id-xyz-abc",
    "item_type":"Resource",
    "user_id":"consumer-xyz-abc"
}'
```

A successful response is as follows:

```
{
"type":"DX-AS-Policy-Success-Resp",
"title":"Policy created successfully"
}
```

On successful validation as shown in Step [19] in Fig. 4, the DX Authorisation Server shall send a successful response as shown above.

ANNEX E

(*Clause* 4)

DX EXAMPLE USE CASES

This annex provides brief descriptions of some use cases that are enabled by the Data Exchange layer using typical data sets available in urban scenarios. A key commonality across the described use case is that these are based on diverse data sets that are typically available with multiple providers/departments/ agencies. Enabling use cases that use such diverse datasets is one of the key benefits of the data exchange layer.

Use Case 1: Solid Waste vehicle routes optimisation	
Description	Based on the waste volume, crowd sourced data from Citizen engagement app, adaptive traffic management information etc, dynamically route the solid waste trucks for operational efficiency and to pick up waste faster for higher citizen satisfaction and cleaner city.
Datasets from Data Exchange	Waste volume data, GPS Locations of the solid waste management vehicles, Adaptive traffic information, Crowd sourced data from Citizen engagement apps, Maps, Navigation data, Bin locations, live/historical bin filling data etc.
Use Case 2: Bus occupancy information with Estimated Time of Arrival (ETA) and fleet optimisations	
Description	Improve the ETA accuracy using the real time traffic info to help the bus operator, such as with inputs for dynamic rescheduling, operational efficiency etc., and also the commuter, such as with inputs on whether to wait for the bus or not, seat availability etc.
Datasets from Data Exchange	Live GPS Locations of all Buses, Routes, Stops, Trip Schedules, Ticket Information between Active stops (AFCS data), Live traffic/congestion information etc.
Use Case 3: Safe and pollution free route to travel in the city	
Description	Based on the city safety parameters, pollutant concentrations grade the safety index for the streets/places and suggest the safe route for the travel
Datasets from Data Exchange	Location wise reported crime data with categories, Surveillance camera feeds, Street light locations/status, Air Quality Monitoring data, Number of people on the streets (Citizen app/Telecom data), Crowd sourced “feeling” data, Maps, Land use data etc.
Use Case 4: Flood Prediction	
Description	Based on the past/present flood sensor data, information about water released from Dams/Lakes, weather data data for rainfall, predict and inform the upcoming flood to the citizens and disaster management authorities.
Datasets from Data Exchange	Flood Sensor data, Video analytics of camera feeds for flood locations,, Citizen App for the GPS coordinates of the users, Telecom data for approximate people in the area, Historical Rainfall/Flooding, Dam/Lake water release data, PA system/Siren/ Sign board locations, Map data including rivers, canals, drains etc.

Use Case 5: Safer places to visit during a pandemic	
Description	Visualize safe routes and places in real-time which can minimize exposure to the virus by scoring each place/route using geospatial data of the positive cases, containment zone polygons, movement and crowd stats using telecom datasets and ward-wise population data. This will help the citizens, administrators and the police to make informed decisions.
Datasets from Data Exchange	Geospatial real-time data of the active cases, containment zone polygons, land use data (offices, schools, markets, shops, malls polygons), crowding and movement of people from telecom data, ward wise population data etc.
Use Case 6: Multimodal transport application	
Description	Takes the data from multiple transport sources, walk paths, Safety indexes, personal preferences and suggests the Best Mode of Transport for the travel against the commuter to open multiple apps for the same.
Datasets from Data Exchange	Rail, Metro, Bus, BRTS, Taxi, RideShare, E-bike, Walk paths, Maps, Routes, Navigation data, Air Quality Monitoring, Safety index of places etc.
Use Case 7: Traffic Analysis and Improvements	
Description	Use the traffic violations/accidents data, adaptive traffic data, other traffic information, road network information and provide comprehensive data based recommendations such as light durations, one way traffic etc.
	Traffic violation data including the location and category of violations, Adaptive traffic information, Traffic accident information, Other Live traffic incident info (such as Tom-Tom data), Junction ITMS camera data, Road network information, Maps etc.

ANNEX F

(*Foreword*)

COMMITTEE COMPOSITION

Smart Infrastructure Sectional Committee, LITD 28

Organization	*Representative(s)*
Indian Institute of Science, Bengaluru	Prof Bharadwaj Amrutur (***Chairman***)
Standardization Testing and Quality Certification (STQC)	Ms Lipika Kaushik
Shrama Technologies Pvt Ltd, Bengaluru	Mr Amarjeet Kumar
Treasure Data	Mr Kumaar Guhan
Amravati Smart City Development Corporation Limited, Mumbai	Mr Siddharth Ganesh
Centre for Development of Telematics, New Delhi	Mr Aurindam Bhattacharya Ms Anupama Chopra (*Alternate*)
Criterion Network Labs, Bengaluru	Mr Jayaprakash Kumar Mr Krishna Kumar Lohati (*Alternate*)
CyanConnode Private Limited, Bengaluru	Mr Manish Widhani Mr Deepak Nimare (*Alternate*)
ERNET India, New Delhi	Mr Paventhan Arumugam
Ericsson India Private Limited, Gurugram	Mr Sendil Kumar Devar
Esri India Technologies Private Limited, Noida	Ms Seema Joshi Vijay kumar (*Alternate*)
Hewlett Packard Enterprise	Mr Devarajan R. Manukumar Nair (*Alternate*)
IEEE India, Bengaluru	Mr Srikanth Chandrasekaran Mr Munir Mohammed (*Alternate*)
India Smart Grid Forum, New Delhi	Mr Reji Kumar Pillai Ms Parul Shribatham (*Alternate*)
Indian Institute of Science, Bengaluru	Mr Vasanth Rajaraman
Intel Technology India Private Limited, Bengaluru	Mr C. Subramanian Mr Anantha Narayanan (*Alternate* I) Mr Sidhartha Mohanty (*Alternate* II)
Ministry of Housing and Urban Affairs, New Delhi	Kunal Kumar Mr Padam Vijay (*Alternate*)
Narnix Technolabs Private Limited, New Delhi	Mr N. Kishor Narang
National Institute of Urban Affairs, New Delhi	Ms Lavanya Nupur
National Smart Grid Mission, Ministry of Power, Gurugram	Mr Arun Misra Smt Kumud Wadhwa (*Alternate* I) Mr Gyan Prakash (*Alternate* II)
PHYTEC Embedded Private Limited, Bengaluru	B. Vallab Rao
Qualcomm India Private Limited, Bengaluru	Dr Vinosh Babu James Dr Punit Rathod (*Alternate*)
Renesas Electronics, Bengaluru	Ravindra Chaturvedi Saurabh Goswami (*Alternate*)

Organization	*Representative(s)*
Schneider Electric's industrial software business - AVEVA, Mumbai	Mr Gourav Kumar Hada
SESEI, New Delhi	Mr Dinesh Chand Sharma
Secure Meters Limited, Gurugram	Mr Uttam Kotdiya Mr Kaustubh Patil (*Alternate* I) Mr Puneet khurana (*Alternate* II) Mr Anil Mehta (*Alternate* III)
Senra Tech Private Limited, New Delhi	Dhiraj Kumar Ankush Kochhar (*Alternate*)
Siemens Limited, Mumbai	Mr Manoj Belgaonkar Mr Ravi Madipadga (*Alternate* I) Mr Pradeep Kapoor (*Alternate* II) Mr Vikram Gandotra (*Alternate* III)
System Level Solutions (India) Private Limited, Anand	Mr Dipen Parmar Mr Foram Modi (*Alternate*)
Tata Consultancy Services Limited, Mumbai	Mr Ramesh Balaji Mr Debashis Mitra (*Alternate*)
Tata Consulting Engineers Limited, Navi Mumbai	Mr Jagdish Shivraj Shige Mr Manoj Kumar (*Alternate*)
Tejas Networks Limited, Bengaluru	Dr Kanwar Jit Singh
Telecommunication Engineering Center, New Delhi	Mr Rajeev Kumar Tyagi Mr Sushil Kumar (*Alternate* I) Mr Uttam Chand (*Alternate* II)
eGovernments Foundation, Bengaluru	Mr Krishnakumar Thiagarajan
In personal capacity	Mr Anish P. K.
In personal capacity	Prof Suptendranath Sarbadhikari
BIS Director General	Ms Reena Garg, Scientist 'F' and Head (Electronics and IT) [Representing Director General (*Ex-officio*)]

Member Secretary

Mr Manikandan K.
Scientist 'D' (Electronics and IT), BIS

Panel involved in the Finalization - LITD 28/P12 Data Exchange Architecture

Organization	*Representative(s)*
IUDX Program Unit, IISc, Bengaluru	Dr Abhay Sharma (***Convener***)
IUDX Program Unit, IISc, Bengaluru	Mr Vasanth Rajaraman
IUDX Program Unit, IISc, Bengaluru	Mr Rakshit Ramesh
IUDX Program Unit, IISc, Bengaluru	Mr Mahidhar Chellamani
IUDX Program Unit, IISc, Bengaluru	Mr Bryan Paul Robert
FIWARE Foundation	Mr Chandra Challagonda
IIIT-Bangalore	Mr Dr Srinath Srinivasa
IIIT-Hyderabad	Ms Anuradha Vattem
Microsoft India Pvt Ltd	Mr Rajesh Kumar
Microsoft India Pvt Ltd	Mr Shalina Bhatia
National Institute of Urban Affairs, New Delhi	Ms Lavanya Nupur
NEC India	Mr Anand Sahu
Redhat India Pvt Ltd	Mr Vinay G Rajagopal
Siemens Limited, Mumbai	Mr Sabishaw Bhaskaran
TCS Research	Sandeep Saxena
Other contributors from IUDX Program Unit, IISc, Bengaluru	Mr C. Subramanian, Mr Anas A. Prof Bharadwaj Amrutur Mr Poorna Chandra Tejasvi Dr Arun Babu Mr Chetan Kumar Mr Kapil Vaswani Mr Anand Lakshmanan

BIBLIOGRAPHY

1. OpenID Connect Core 1.0, Available at https://openid.net/specs/openid-connect-core-1_0.html
2. IETF RFC 6749: The OAuth 2.0 Authorization Framework. Available at https://tools.ietf.org/html/rfc6749
3. IETF RFC 6749: The OAuth 2.0 Authorization Framework: The Bearer Token Usage. Available at: https://tools.ietf.org/html/rfc6750.
4. ETF RFC 7519: JSON Web Token (JWT). Available at: https://tools.ietf.org/html/rfc7519
5. MQTT 5.0, OASIS Standard. Available at https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html
6. Advanced Message Queuing Protocol (AMQP): v0.9.1. Available at: https://www.amqp.org/specification/0-9-1/amqp-org-download
7. JSON Schema. Available at: https://json-schema.org/
8. IETF RFC 7396: "JSON Merge Patch". Available at https://tools.ietf.org/html/rfc7396.
9. Linked Data. Tim Berners-Lee. Personal View, imperfect but published. Available at: http://www.w3.org/DesignIssues/LinkedData.html
10. IETF RFC 7522: Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants. Available at: https://tools.ietf.org/html/rfc7522
11. IETF RFC 3987: Internationalized Resource Identifiers (IRIs). Available at: https://tools.ietf.org/html/rfc3987
12. OWASP API Security Project. Available at: https://owasp.org/www-project-api-security/
13. NIST SP 800-204, Security Strategies for Micro-services based application systems. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf
14. OpenAPI Specification. Available at: https://swagger.io/specification/

This page has been intentionally left blank

Bureau of Indian Standards

BIS is a statutory institution established under the *Bureau of Indian Standards Act*, 2016 to promote harmonious development of the activities of standardization, marking and quality certification of goods and attending to connected matters in the country.

Review of Indian Standards

Amendments are issued to standards as the need arises on the basis of comments. Standards are also reviewed periodically; a standard along with amendments is reaffirmed when such review indicates that no changes are needed; if the review indicates that changes are needed, it is taken up for revision. Users of Indian Standards should ascertain that they are in possession of the latest amendments or edition by referring to the latest issue of 'BIS Catalogue' and 'Standards: Monthly Additions'.

This Indian Standard has been developed from Doc No.: LITD 28 (17249).

Amendments Issued Since Publication

Amend No.	Date of Issue	Text Affected

BUREAU OF INDIAN STANDARDS

Headquarters:

Manak Bhavan, 9 Bahadur Shah Zafar Marg, New Delhi 110002
Telephones: 2323 0131, 2323 3375, 2323 9402 *Website*: www.bis.gov.in

Regional Offices:		*Telephones*
Central	: Manak Bhavan, 9 Bahadur Shah Zafar Marg, NEW DELHI 110002	2323 7617, 2323 3841
Eastern	: 1/14 C.I.T. Scheme VII M, V.I.P. Road, Kankurgachi, KOLKATA 700054	2337 8499, 2337 8561, 2337 8626, 2337 9120
Northern	: Plot No. 4-A, Sector 27-B, Madhya Marg, CHANDIGARH 160019	265 0206, 265 0290
Southern	: C.I.T. Campus, IV Cross Road, CHENNAI 600113	2254 1216, 2254 1442, 2254 2519, 2254 2315
Western	: Manakalaya, E9 MIDC, Marol, Andheri (East), MUMBAI 400093	2832 9295, 2832 7858, 2832 7891, 2832 7892

Branches : AHMEDABAD. BENGALURU. BHOPAL. BHUBANESHWAR. COIMBATORE. DEHRADUN. DURGAPUR. FARIDABAD. GHAZIABAD. GUWAHATI. HYDERABAD. JAIPUR. JAMMU. JAMSHEDPUR. KOCHI. LUCKNOW. NAGPUR. PARWANOO. PATNA. PUNE. RAIPUR. RAJKOT. VISAKHAPATNAM.

Published by BIS, New Delhi