**Problem**

There is no standard syntax for representing tags from multiple ontologies for Semantic Web applications.

# BACnet RDF **Tags**

This document proposes a new annex to ASHRAE/ANSI Standard 135-2016 for new syntax and interpretation of the name and value components of a BACnetNameValue sequence.

The interpretation is designed to closely follow the RDF model for Linked Data and makes the content simple to interpret by adopting syntax conventions and algorithms from the Turtle and JSON-LD RDF serialization formats which are already well established.

References to Clause Y.1.4 in Clause 12 for string values for the names of tags are replaced by references to this annex. For example, an updated Clause 12.2.38:

12.2.38 Tags

This property, of type BACnetARRAY of BACnetNameValue, is a collection of tags for the object. See Clause JJ for restrictions on the string values used for the names of these tags and for a description of tagging and the mechanism by which tags are defined.

This began as an update of Clause Y.1.4 however the resulting changes use completely different terminology and most of ANNEX Y - ABSTRACT DATA MODEL presumes an XML Information Set model of data. The RDF data model used for Linked Data is significantly different.

Creating an RDF or Linked Data model that delivers the same essential knowledge that is described and encoded in data and metadata sections of ANNEX Y is future work.

# ANNEX JJ - LINKED DATA MODEL (NORMATIVE)

(This annex is part of this standard and is required for its use.)

This annex defines the syntax and interpretation of the name and value components of a BACnetNameValue sequence for defining, transmitting, and storing semantic information for building management and business applications using the [Resource Description Framework (RDF)](https://www.w3.org/RDF/) defined by the [World Wide Web Consortium (W3C)](https://www.w3.org/).

## JJ.1 Introduction

The Resource Description Framework (RDF) is a framework for expressing information about resources in a machine readable form. A resource can be anything, including physical objects like fans and pumps, or abstract concepts like processes. The framework defines a way to make statements that describe resources and relationships between them.

RDF can be used to publish and interlink data on the Web. For example, retrieving http://www.example.org/vav-13 could provide data about a VAV box, including the fact that it is located in a specific room identified by its International Resource Identifier (IRI). Retrieving the IRI for the room could then provide more data about it, including its structural design, expected use as an office or laboratory, and links to other datasets that contain environmental conditions like temperature and humidity.

### JJ.1.1 RDF Model Terminology

And RDF model is a series of RDF statements, or more commonly called an RDF triple. A RDF triple has three constituent parts: the subject, predicate, and object of the statement. The subject and predicate of a statement are IRIs, the object may be an IRI or a literal value.

The syntax for IRI References and IRIs is defined in [RFC3987](https://tools.ietf.org/html/rfc3987" \l "section-2.2).

#### JJ.1.1.1 Literal Values

Literal values are similar to fundamental datatype values in BACnet such as booleans, integers, and strings. They are encoded in text based serialization formats by the same rules as used in [XML Schema Datatypes](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/" \l "section-Datatypes).

When an RDF statement describes a relationship between a resource and a literal value, the statement closely resembles the [Entity-Attribute-Value (EAV)](https://en.wikipedia.org/wiki/Entity–attribute–value_model) in [object-oriented design](https://en.wikipedia.org/wiki/Object-oriented_design).

RDF does not allow statements about literal values, so it is invalid to have a statement that uses the number "12" in the subject part. It also does not allow statements with literals as a predicate.

String literals can optionally be associated with a language tag which is used for presenting content to a user.

#### JJ.1.1.2 Subjects and Objects

When referring to a specific RDF triple, the terms "subject", "predicate" and "object" are used for their individual parts. However, when describing a statement in the context of a model or a collection of statements, the subject and object portions are collectively referred to as objects and the relationship is referred to as property.

For example, assume that http://www.example.org/vav-13 is the IRI for a VAV box, http://demo.org/sample#serves is the IRI for a generic relationship between a thing and what it serves, and http://www.example.org/room-303 is the IRI for a specific room. The RDF statment representing "VAV-13 serves Room 303" (with those three IRIs in that order) describes a relationship between a VAV object and a room object.

#### JJ.1.1.3 Blank Nodes

IRIs and literals together provide the basic material for creating RDF statements. In addition, it is sometimes handy to be able to talk about resources without using a global identifier. For example, in modeling the flow or air in the duct work between an air conditioning unit and multiple VAV boxes, it would be advantageous to reference the junctions between ducts without requiring a global identifier.

Blank node identifiers are local identifiers that are used in some concrete RDF syntaxes or RDF store implementations. They are always locally scoped to the file or RDF store, and are not persistent or portable identifiers for blank nodes. Blank node identifiers are not part of the RDF abstract syntax, but are entirely dependent on the concrete syntax or implementation.

The blank node identifier format in this specification format is the same that is used in other common concrete RDF syntaxes with the "\_:" prefix, and suggests an object identifier format to make it easier to understand for some users assuming that many references in tag sets will be between BACnet objects.

### JJ.1.2 Comparing and contrasting XML

While RDF superficially shares may of the same goals as XML by being machine readable, it differs from XML in important ways; RDF is a more flexible graph structured data model, it uses IRIs as global unique identifiers, and RDF models are themselves expressed in RDF.

#### JJ.1.2.1 XML and RDF Data Models

XML is a context free syntax for exchanging content which means that there are no assumptions made about the kind of content being exchanged between two applications or services other than that it is tree structured. For two applications to agree on the meaning of a document and its component pieces they must also agree on a specific schema for its interpretation.

The XML syntax in Annex Q describes a specific subset of XML that is based on the data model in Annex Y, the two together provide a mechanism for exchanging content.

The RDF Data Model has no specific syntax, there are a number of text based serialization formats that can be used to exchange content. For example, the RDF/XML format specifies an XML schema for systems where there is an existing XML processing infrastructure already established, the N-Triples format is the simplest but also the most verbose, the Turtle format is designed to be readily understood by both humans and applications, and the JSON-LD format is designed to assist application developers with their existing JSON tools to present content in RDF.

#### JJ.1.2.2 XML and RDF Identifiers

XML Documents have no mechanism for identifying resources, there is a separate XPath specification for referencing nodes or node sets within an XML document and XLink and XPointer are additional specifications for linking between documents. Annex W uses these path expressions, along with specific paths to help with resource discovery.

Resource identification is at the core of RDF and uses IRIs for both the identifier of a "thing" such as a pump and for the properties that describe the "thing" such as its size or capacity. The URLs (Uniform Resource Locators) that are used as Web addresses are one form of IRI. Other forms of IRI provide an identifier for a resource without implying its location or how to access it. The notion of IRI is a generalization of URI (Uniform Resource Identifier), allowing non-ASCII characters to be used in the IRI character string. IRIs are specified in [RFC 3987](https://tools.ietf.org/html/rfc3987).

#### JJ.1.2.3 XML Schemas and RDF Ontologies

XML documents and their associated schemas are closely coupled, changes in the structure of an XML document must be accupanied by corresponding changes to the schema. The applications at both the source and destination of a document exchange must agree on the updated schema. Because there is no identifier for a resource apart from its path, changes to a schema that involve moving the information associated with a resource to another location within the document can break relationships between resources.

XML document structures are described using [Document Type Definitions (DTD)](https://www.iso.org/standard/41009.html), [RELAX NG](https://relaxng.org/), or [XML Schema](https://www.w3.org/standards/xml/schema) documents, each of which have their own syntax and content organization.

RDF is designed for interlinking various datasets within an organization, enabling cross-dataset queries by cooperating on IRIs, independant of the document format that is used to exchange organization model data. Information about a resource can be distributed among multiple datasets.

An ontology in the context of RDF is a set of concepts and categories in a subject area or domain that describes resources by describing what resources exist or may be said to exist and how they may be grouped, related within a heirarchy, and subdivided according to their similarities and differences.

RDF models are described using one or more of the many content organization ontologies that are themselves described in RDF. Query applications and knowledge management tools can validate models, for example making sure that two sets of definitions do not conflict with respect to principles of First Order Logic, and use both the model data and the instance data at the same time.

## JJ.2 Terminology

The terminology described in this section is used throughout this Annex to describe specific components and conceptual collections of components.

A tag name is the name component of a BACnetNameValue sequence, a tag value is the value component.

A tag set is the collection of all of the BACnetNameValue array elements in the Tags property of an object, or those in a single array element of the Subordinate\_Tags property of a Structured View Object. While the former is an array and the latter is a list, they are both a tag set.

The contents of a two tags sets are considered equal if they contain identical elements, tag set content is not dependent on the order of the BACnetNameValue component elements.

The expression tag set of an object is the tag set that is encoded in the Tags property of the object. The subordinate tag set of an object is the object referenced by the Subordinate\_List with the content of the list with the matching index in the Subordinate\_Tags.

The tag set for an object only refers to the BACnetNameValue elements of the Tags property. There may be more then one subordinate tag set for the same object in one or more structured view objects in the same device, or in other devices in the BACnet intranet. The tag set for an object and the subordinate tag sets for the object may represent identical statements, or conflicting statements according to some model.

A BACnet object contains a tag when there is at least one BACnetNameValue element with a matching tag name in the tag set of the object.

The graph for an object is the RDF graph built from the set of RDF statements from its tag set. Similarly, the subordinate graph for an object is the RDF graph built from the set of RDF statements from its subordinate tag set. The graph of RDF statements follow the [open world assumption](https://en.wikipedia.org/wiki/Open-world_assumption) which implies that there is no one universal collection of statements.

The graph for a device is the union of all the graphs for all of the objects in the device, which may include graphs for objects in other devices, and the optional RDF document(s) contained in the XDD file of a device.

A vocabulary is a common IRI prefix for properties and types.

### JJ.2.1 Default Vocabulary

Tag names are resolved into IRIs according to the processing algorithm in Clause JJ.3.3 and are expected to follow [RFC7320 URI Design and Ownership](https://tools.ietf.org/html/rfc7320). The default vocabulary prefix is the URL https://data.ashrae.org/bacnet.

## JJ.3 Tag Names

Tag names are directives, prefix declarations, local names, prefixed names, or IRI references. Tag names shall not be an empty string.

Directives are tag names that begin with "@" and contain interpretation information. All directive tag names are reserved.

Prefix declarations associate a prefix label with a vocabulary IRI. Prefixed names are a prefix and a local part separated by a colon ":". These are referred to in the JSON-LD specification as a Compact IRI and in the Turtle specification as a prefixed name .

Local names are names without a prefix.

### JJ.3.1 Directives

Directives provide context information for establishing an identity for the subject of tag set, interpreting tag names and translating them to IRIs for the predicate of an RDF triple, and setting a language for descriptive string literal values when they are the object of an RDF triple.

#### JJ.3.1.1 @id Directive

The @id directive defines the IRI of the subject for the tag set. The tag value is a Character String which may be an IRI, prefixed name, local name, or blank node.

There can be only one @id tag in a tag set. If the tag set does not contain an @id tag the subject IRI is the blank node \_:T,N which is the blank node prefix \_: followed by the object identifier of the object of the tag set.

T is either a standard type name exactly equal to the names specified in the definition for BACnetObjectTypes in Clause 21 or a decimal number with no leading zeroes. N represents the object instance number and is a decimal number with no leading zeroes. See JJ.5.6.4.

Note that a period is used to separate the type and instance number, rather than a comma used in Clause Y.20.1.

The two syntax forms, where one uses the enumerated value of the object type and the other that uses the enumeration name, cannot be used in the same device graph to refer to one object because different blank node identifiers always reference different objects in an RDF graph.

#### JJ.3.1.2 @base Directive

The @base directive defines the Base IRI used to resolve relative IRIs per RFC3986 section 5.1.1, "Base URI Embedded in Content". The tag value is a Character String and must be an IRI.

There can be only one @base tag in a tag set. If the tag set does not contain a @base tag, the tag set inherits the @base tag value in the tag set of the device object, if one is provided.

#### JJ.3.1.3 @vocab Directive

The @vocab directive is used to expand properties and values with a common prefix IRI. See Clause JJ.3.

There can be only one @vocab tag in a tag set. If the tag set does not contain a @vocab tag, the tag set inherits the @vocab tag value in the tag set of the device object, if one is provided.

#### JJ.3.1.4 @language Directive

The default language is set in the tag set using the @language tag name whose value is a Character String with a [BCP47](https://tools.ietf.org/html/bcp47) language code.

There can be only one @language tag in a tag set. If the tag set does not contain a @language tag, the tag set inherits the @language tag value in the tag set of the device object, if one is provided.

If there is no @language tag in an object tag set or in the tag set of the device object, the character strings values are plain strings.

### JJ.3.2 Prefix Label Declaration

A prefix declaration is used for prefixed names and is a local name terminated with a colon ':'. The tag value must be a Character String and must be an IRI.

There can only be one prefix label declaration in a tag set for any given prefix. Prefix declarations are inherited from the device object tag set.

### JJ.3.3 Local Names

A tag name may be a local name that matches the isegment-nz-nc production as defined in [RFC3987](https://tools.ietf.org/html/rfc3987). Local names are expanded using the vocabulary and base IRI defintions, see Clause JJ.5.5.1.

### JJ.3.4 Prefixed Names

A tag name is in a compact IRI represented as a prefix:suffix combination, the prefix matches a prefix label declaration within the active context, and the suffix does not begin with two slashes (//).

If the prefix is not defined in the active context, or the suffix begins with two slashes (such as in http://example.com), the value is interpreted as absolute IRI. If the prefix is an underscore (\_), the value is interpreted as blank node identifier.

### JJ.3.5 IRI References

The tag name may be an IRI.

## JJ.4 Tag Values

Tag values are limited to primitive data type values and BACnetDateTime values.

## JJ.5 Tag Set Conversion Procedure

This clause specifies the process that is used to construct a tag set for an object to the graph of the object. The result of this procedure is that all directives are removed, all terms and compact IRIs are expanded to absolute IRIs, blank node identifiers, or keywords and all values are expressed as RDF literal values.

### JJ.5.1 Overview

The algorithm creates a context, a collection of directives and prefix information used to resolve names and then translates each of the BACnetNameValue elements to an RDF triple.

### JJ.5.2 Context Constuction

Begin with an empty context. Select from the device object tag set the @base, @language, @vocab directives, and the prefix label declarations and save them in the context.

From the object tag set, or the subordinate tag set, select the @base, @language, and @vacab directives, if they exist, and merge them into the context, overwriting the existing context values is there are any. Similarly, select the prefix label declarations and merge them as well, replacing any existing declarations with an identical prefix.

### JJ.5.3 IRI Expansion

IRI expansion may occur during content processing of the value of the @id directive, the local names, or the prefixed names of the BACnetNameValue elements.

#### JJ.5.3.1 Base IRI

The @base directive defines the Base IRI used to resolve relative IRIs per [RFC3986 Section 5.1.1, "Base URI Embedded in Content"](https://tools.ietf.org/html/rfc3986" \l "section-5.1.1). The base IRI must be an absolute IRI. If the @base directive is omitted, the current vocabulary is the default vocabulary, see Clause JJ.1.2.

#### JJ.5.3.2 Vocabulary IRI

The @vocab directive is an absolute or relative IRI. If the directive value is an absolute IRI then it becomes the current vocabulary. If the directive value is a relative IRI is it resolved relative to the current vocabulary per [RFC3986 Section 5.2, "Relative Resolution"](https://tools.ietf.org/html/rfc3986" \l "section-5.2) and becomes the new vocabulary.

If the @base and vocab directives are absent, the vocabulary is the default vocabulary, see Clause JJ.1.2.

#### JJ.5.3.3 Prefix Expansion

Each of the prefix declarations in the context creates a vocabulary that is used to resolve prefixed names. The prefix value is an absolute or relative IRI. If the prefix value is an absolute IRI then it becomes the vocabulary. If the value is a relative IRI it is resolved relative to the current vocabulary, as created by processing both the base IRI in Clause JJ.5.3.1 and vocabulary IRI in Clause JJ.5.3.2, per [RFC3986 Section 5.2, "Relative Resolution"](https://tools.ietf.org/html/rfc3986" \l "section-5.2).

### JJ.5.4 Subject IRI

From the object tag set the @id directive if it exists and resolve the value according to the algorithm described in Clause JJ.3.3. If the @id is not present, generate a blank node identifier using the object identifier of the object, the blank node prefix "\_:" and the algorithm described in Clause JJ.5.6.7.

The value of the @id directive may be an absolute IRI, relative IRI, prefixed name or local name. The value is expanded by the algorithm in Clause JJ.5.5.

### JJ.5.5 Tag Name Expansion

The tag name may be a local name, prefixed name, or IRI reference.

#### JJ.5.5.1 Local Name Expansion

The IRI for a local name is the value of the current vocabulary, as created by processing both the base IRI in Clause JJ.5.3.1 and vocabulary IRI in Clause JJ.5.3.2, concatenated with the tag name.

#### JJ.5.5.2 Prefixed Name Expansion

The prefixed name is split into a prefix and a suffix at the first occurence of a colon (:).

If the prefix is not defined in the active context, or the suffix begins with two slashes (such as in http://example.com), the value is interpreted as absolute IRI. If the prefix is an underscore (\_), the value is interpreted as blank node identifier.

Otherwise, the IRI for a prefixed name is the value of the prefix expansion, as created by Clause JJ.5.3.3, concatenated with the suffix.

#### JJ.5.5.3 IRI Reference Expansion

If the tag name is an IRI reference, it is resolved relative to the current vocabulary, as created by processing both the base IRI in Clause JJ.5.3.1 and vocabulary IRI in Clause JJ.5.3.2, per [RFC3986 Section 5.2, "Relative Resolution"](https://tools.ietf.org/html/rfc3986" \l "section-5.2).

### JJ.5.6 Tag Value Conversion

Tag values are limited to primitive datatypes and BACnetDateTime. There is no mechanism for specifying the RDF datatype or multiple languages for literals within the tag set for an object, if that information is to be expressed then it is put in the RDF document in the XDD file for the device.

Each data value shall be represented as a string that is appropriate for the data type, and shall be formatted as if returned in 'plain text' from the services described in Clause W.9. The following table summarizes the serialization format for each data type and its reference to [XML Schema Datatypes](https://www.w3.org/TR/xmlschema-2/).

| Source Data Type | Serialization Type | Examples | Notes |
| --- | --- | --- | --- |
| BitString | xsd:string | fault  fault;overridden | A semicolon separated list of the names of the bits that are true. i.e., an empty string means that all bits are false. See Clause Y.12.11 |
| Boolean | xsd:boolean | true  false |  |
| CharacterString | xsd:string | Hello, world! | See Clause JJ.5.6.1 |
| Date | xsd:date | 2018-01-24 |  |
| DatePattern | bacnet:datePattern | \*-01-24  2018-01-\*  \*-\*-\* | See Clause Y.12.14 |
| DateTime | xsd:dateTime | 2018-01-24T08:56:00+01:00  2018-01-24T07:56:00.00Z |  |
| DateTimePattern | bacnet:dateTimePattern | 2018-01-24 10:\*:\*.\*  \*-\*-\* 3 10:00:00.00  \*-\*-\* \*:\*:\*.\* | See Clause Y.12.16 |
| Double | xsd:double | 123456789.00 |  |
| Enumerated | xsd:string | high  idle | See Clause Y.12.12 |
| Integer | xsd:integer | 1234 |  |
| Null |  |  | See Clause JJ.5.6.2 |
| Link | xsd:string | <http://example.org/abc> | See Clause JJ.5.6.3 |
| ObjectIdentifier | xsd:string | calendar,12 | See Clause JJ.5.6.4 |
| OctetString | xsd:hexBinary | 0103CAFEBABE99  0103cafebabe99 |  |
| Real | xsd:float | 1234.56 |  |
| Time | xsd:time | 12:05:22  12:05:22.55 | Fractional seconds is optional |
| TimePattern | bacnet:timePattern | 10:24:\*.\*  \*:\*:\*.\* | See Clause Y.12.18 |
| Unsigned | xsd:nonNegativeInteger | 1234 |  |

#### JJ.5.6.1 CharacterString Value

Character string content may be an IRI reference if the content is enclosed in angle-brackets '<' and '>'. See Clause JJ.5.6.3.

Character string content that is not an IRI reference may be a [language-tagged string](https://www.w3.org/TR/rdf11-concepts/" \l "dfn-language-tagged-string) if the @language directive is provided in the tag set. See Clause JJ.3.1.4.

#### JJ.5.6.2 Null Value

Null values are used as an abbreviation for saying an object is an instance of an RDF class. The tag name becomes the object of the RDF statement and the predicate becomes http://www.w3.org/1999/02/22-rdf-syntax-ns#type which is commonly represented by the prefix name rdf:type, or in the case of Turtle, the a keyword.

#### JJ.5.6.3 Link Value

Links are IRI references and are CharacterString values with content that is enclosed in angle-brackets '<' and '>'. The content may be an absolute IRI, relative IRI, prefixed name or local name. The value is expanded by the algorithm in Clause JJ.5.5.

#### JJ.5.6.4 ObjectIdentifier Values

Object identifiers are character strings encoded in the form T,N, where T is either a standard type name exactly equal to the names specified in the definition for BACnetObjectTypes in Clause 21 or a decimal number with no leading zeroes. N represents the object instance number and is a decimal number with no leading zeroes.

# ANNEX JK - BACnetNameValue EXAMPLES (INFORMATIVE)

(This annex is not part of this standard but is included for informative purposes only.)

This annex provides examples of the use of Tags property values and how those values are transliterated into RDF statements by showing the same content using Turtle.

## JK.1 Example of a single local tag in the ASHRAE vocabulary

The following partial object definition is an object with a single tag named "temp" and a NULL value. The blank node identifier uses the unsigned decimal number form for the object type.

Object Identifier = (Analog Value, 1)

Tags = (("temp", NULL))

Turtle:

@prefix ashrae: <http://data.ashrae.org/223/2019#> .

[] a ashrae:temp .

JSON-LD:

{

"@context": {

"@vocab": "http://data.ashrae.org/223/2019#"

},

"@id": "\_:N1e46336747b140139b51be57eeeb7462",

"@type": "temp"

}

## JK.2 Example of a single local tag in an alternate vocabulary

The following partial object definition changes the base prefix to some contains the shortest possible prefix name declaration and use. It is an object with a single tag named ":temp" and a NULL value.

Object Identifier = (Analog Value, 1)

Tags = (("@vocab", "http://example.org/"), ("temp", NULL))

Turtle

@prefix ns1: <http://example.org/> .

[] a ns1:temp .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N21f6d33507424f3ba4ad91b4449c915c",

"@type": "temp"

}

## JK.3 Example of a tag with a short prefix

The following partial object definition contains the shortest possible prefix name declaration and use. It is an object with a single tag named ":temp" and a NULL value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":sensor", NULL))

Turtle

@prefix : <http://example.org/> .

[] a :sensor .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N1936af9abd334a5fb39045147ad31b39",

"@type": "sensor"

}

## JK.4 Example of a tag with boolean value

The following partial object definition contains a prefix name declaration and use, a single tag named ":inhibit-delay" and with a TRUE value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":inhibit-delay", TRUE))

Turtle

@prefix : <http://example.org/> .

[] :inhibit-delay true .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:Nf4352f9d4bc84f6d9b8f70063f1e7a32",

"inhibit-delay": true

}

## JK.5 Example of a tag with an unsigned integer value

The following partial object definition contains a prefix name declaration and use, a single tag named ":inhibit-delay" and with an unsigned integer:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":time-duration", 12))

Turtle

@prefix : <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

\_:analog-value.1 :time-duration "12"^^xsd:nonNegativeInteger .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N7e4520737bb94371aefa4b5db3836eab",

"time-duration": {

"@type": "http://www.w3.org/2001/XMLSchema#nonNegativeInteger",

"@value": "12"

}

}

This JSON-LD serialization contains a context that defined the time-duration to be a non-negative integer so the value is simpler. It is common to provide the context information in a separate document, reducing the size of the JSON-LD document.

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/",

"time-duration": {

"@type": "http://www.w3.org/2001/XMLSchema#nonNegativeInteger"

}

},

"@id": "\_:N7caa3a281d61413981e2caf37dc3408e",

"time-duration": "12"

}

## JK.6 Example of tags with a signed integer value

The following partial object definition contains a prefix name declaration and use, and a pair of tags named ":high-offset" and ":low-offset" with signed integer values:

Object Identifier = (Analog Value, 1)

Tags = ((":", "<http://example.org/>"),  
 (":high-offset", 10), (":low-offset", -15))

Turtle

@prefix : <http://example.org/> .

\_:analog-value.1  
 :high-offset 10 ;  
 :low-offset -15 .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N3e5d399f812e4e469798ac45cea10003",

"high-offset": 10,

"low-offset": -15

}

## JK.7 Example of tags with a Real value

The following partial object definition contains a prefix name declaration and use, a single tag named ":inhibit-delay" and with a single precision floating point value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":deadband", 0.5))

Turtle

@prefix : <http://example.org/> .

\_:analog-value.1 :deadband "0.5"^^xsd:float .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N21bb05cfd66f4330a68f9436b9582c2e",

"deadband": {

"@type": "http://www.w3.org/2001/XMLSchema#float",

"@value": "0.5"

}

}

## JK.8 Example of tags with a Double value

The following partial object definition contains a prefix name declaration and use, a single tag named ":parsecs" and with a double precision floating point value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":parsecs", 12.5))

Turtle

@prefix : <http://example.org/> .

[] :parsecs 1.25e+01 .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N20b046a915ee49fe8dda0ebf4af005ab",

"parsecs": 12.5

}

## JK.9 Example of tags with an octet string value

The following partial object definition contains a prefix name declaration and use, a single tag named ":key" and with an octet string, length 2:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":key", X"0123"))

Turtle

@prefix : <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

\_:analog-value.1 :key "MDEyMw=="^^xsd:base64Binary .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N8b1f2e3836984a48b41c71ba67724bee",

"key": {

"@type": "http://www.w3.org/2001/XMLSchema#base64Binary",

"@value": "MDEyMw=="

}

}

## JK.10 Example of tags with a character string value

The following partial object definition contains a prefix name declaration and use, a single tag named ":welcome" and with a simple string:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"), (":welcome", "Hello, world!"))

Turtle

@prefix : <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

\_:analog-value.1 :welcome "Hello, world!" .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N20bfaa33c2fa4a6bad7a70c0ff7ae897",

"welcome": "Hello, world!"

}

## JK.11 Example of tags with a bit string value

The following partial object definition contains a prefix name declaration and use, a single tag named ":statusFlags" and with a bit string, length 5:

Object Identifier = (Analog Value, 1)

Tags = (":event-enable", [1, 0, 1])

Turtle

@prefix BACnet: <http://data.ashrae.org/bacnet/2016#> .

[] BACnet:event-enable "to-offnormal;to-normal" .

JSON-LD:

{

"@context": {

"@vocab": "http://data.ashrae.org/bacnet/2016#"

},

"@id": "\_:N5053df34261b4997aba5eec3df1cdf1e",

"event-enable": "to-offnormal;to-normal"

}

## JK.12 Example of tags with an enumerated value

The following partial object definition contains a single tag named "units" defined in the default vocabulary and with an enumerated value, also defined in the default vocabulary:

Object Identifier = (Analog Value, 1)

Tags = (("units", Enumerated(63))) -- Degrees Kelvin

Turtle

@prefix ashrae: <http://data.ashrae.org/223/2019#> .

[] ashrae:units "degrees-kelvin" .

JSON-LD:

{

"@context": {

"@vocab": "http://data.ashrae.org/223/2019#"

},

"@id": "\_:N5c427f64d19746819e0eb94ed68a50aa",

"units": "degrees-kelvin"

}

This is an example where the units are referencing a specific 223 concept that can then be mapped to QUDT in the ASHRAE ontology.

Object Identifier = (Analog Value, 1)

Tags = (("units", "<degrees-kelvin>")) -- Degrees Kelvin

Turtle

@prefix ashrae: <http://data.ashrae.org/223/2019#> .

[] ashrae:units ashrae:degrees-kelvin .

JSON-LD:

{

"@context": {

"@vocab": "http://data.ashrae.org/223/2019#"

},

"@id": "\_:Nf0b3fdf5a50d4833837aca438fb6bc85",

"units": {

"@id": "http://data.ashrae.org/223/2019#degrees-kelvin"

}

}

## JK.13 Example of tags with a date value

The following partial object definition contains a prefix name declaration and use, a single tag named ":startDate" and with a date value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "<http://example.org/>"),  
 (":startDate", Date(2019, 7, 9))

Turtle

@prefix : <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] :startDate "2019-07-09"^^xsd:date .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N87b451afe0464cd1b6dec3a4a40c9982",

"startDate": {

"@type": "http://www.w3.org/2001/XMLSchema#date",

"@value": "2019-07-09"

}

}

## JK.14 Example of tags with a time value

The following partial object definition contains a prefix name declaration and use, a single tag named ":startTime" and with a time value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"),

(":startTime", Time(10, 19, 15, 20))

)

Turtle

@prefix : <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] :startTime "10:19:15.20"^^xsd:time .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N48cd4efdeb2e4189a9feba01eb488567",

"startTime": {

"@type": "http://www.w3.org/2001/XMLSchema#time",

"@value": "10:19:15.20"

}

}

## JK.15 Example of tags with a datetime value

The following partial object definition contains a prefix name declaration and use, a single tag named ":opening" and with a datetime value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "http://example.org/"),

(":opening", (Date(2019, 7, 9), Time(10, 19, 15, 20)))

)

Turtle

@prefix : <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] :opening "2019-07-09T10:19:15.20"^^xsd:dateTime .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N649b98a3bb1a4d729e0cb51a3966da88",

"opening": {

"@type": "http://www.w3.org/2001/XMLSchema#dateTime",

"@value": "2019-07-09T10:19:15.20"

}

}

## JK.16 Example of tags with an object identifier value

The following partial object definition contains a prefix name declaration and use, a single tag named ":spid" and with an object identifier value:

Object Identifier = (Analog Value, 1)

Tags = ((":", "<http://example.org/>"),  
 (":spid", ObjectIdentifier('analog-value' , 3)))

Turtle

@prefix : <http://example.org/> .

[] :spid "analog-value,3" .

JSON-LD:

{

"@context": {

"@vocab": "http://example.org/"

},

"@id": "\_:N59ac2aa617fd430cbed5ddd324a9c530",

"spid": "analog-value,3"

}