Stream:	Internet Engineering Task Force (IETF)		
RFC:	9880		
Category:	Standards Track		
Published:	January 2026		
ISSN:	2070-1721		
Authors:	M. Koster, Ed.	C. Bormann, Ed.	A. Keränen
	KTC Control AB	*Universität Bremen TZI*	*Ericsson*

RFC 9880
Semantic Definition Format (SDF) for Data and Interactions of Things

Abstract

The Semantic Definition Format (SDF) is a format for domain experts to use in the creation and maintenance of data and interaction models that describe Things, i.e., physical objects that are available for interaction over a network. An SDF specification describes definitions of SDF Objects/SDF Things and their associated interactions (Events, Actions, and Properties), as well as the Data types for the information exchanged in those interactions. Tools convert this format to database formats and other serializations as needed.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9880.

Copyright Notice

Table of Contents

1. Introduction

The Semantic Definition Format (SDF) is concerned with Things, namely physical objects that are available for interaction over a network. SDF is a format for domain experts to use in the creation and maintenance of data and interaction models that describe Things. An SDF specification describes definitions of SDF Objects/SDF Things and their associated interactions

(Events, Actions, and Properties), as well as the Data types for the information exchanged in those interactions. Tools convert this format to database formats and other serializations as needed.

SDF is designed to be an extensible format. The present document constitutes the base specification for SDF, "base SDF" for short. In addition, SDF extensions can be defined, some of which may make use of extension points specifically defined for this in base SDF. One area for such extensions would be refinements of SDF's abstract interaction models into protocol bindings for specific ecosystems (e.g., [SDF-MAPPING]). For the use of certain other extensions, it may be necessary to indicate in the SDF document using them that a specific extension is in effect; see Section 3.1 for details of the `features` quality that can be used for such indications. With extension points and feature indications available, base SDF does not define a "version" concept for the SDF format itself (as opposed to version indications within SDF documents indicating their own evolution; see Section 3.1).

1.1. Structure of This Document

After introductory material and an overview (Section 2) over the elements of the model and the different kinds of names used, Section 3 introduces the main components of an SDF model. Section 4 revisits names and structures them into namespaces. Section 5 discusses the inner structure of the Objects defined by SDF, the sdfObjects, in further detail. Section 6 discusses how SDF supports composition. Conventional Sections (IANA Considerations, Security Considerations, Normative References, and Informative References) follow. The normative Appendix A defines the syntax of SDF in terms of its JSON structures, employing the Concise Data Definition Language (CDDL) [RFC8610]. This is followed by the informative Appendix B, a rendition of the SDF syntax in a "JSON Schema" format as they are defined by `json-schema.org` (collectively called JSO). The normative Appendix C defines certain terms ("data qualities") used at the SDF data model level that were inspired by JSO. The informative Appendix D provides a few examples for the use of composition in SDF. Finally, Appendix E provides some historical information that can be useful in upgrading earlier, pre-standard SDF models and implementations to base SDF.

1.2. Terminology and Conventions

Terms introduced in this section are capitalized when used in this section. To maintain readability, capitalization is only used when needed where they are used in the body of this document.

1.2.1. Programming Platform Terms

The following definitions mention terms that are used with specific meanings in various programming platforms, but often have an independent definition for this document, which can be found further below in this section.

Element: A generic term used here in its English sense. Exceptionally, in Appendix C, the term is used explicitly in accordance with its meaning in the JSON ecosystem, i.e., the elements of JSON arrays.

Entry: A key-value pair in a map. (In JSON maps, sometimes also called "member".)

Map: A collection of entries (key-value pairs) where there are no two entries with equivalent keys. (Also known as associative array, dictionary, or symbol table.)

Object: An otherwise very generic term that JavaScript (and thus JSON) uses for the kind of maps that were part of the original languages from the outset. In this document, Object is used exclusively in its general English meaning or as the colloquial shorthand for sdfObject, even if the type name "`object`" is imported with JSON-related semantics from a data definition language.

Property: Certain environments use the term "property" for a JSON concept that JSON calls "member" and is called "entry" here, or sometimes just for the map key of these. In this document, the term Property is specifically reserved for a certain kind of Affordance, even if the map key "`properties`" is imported with JSON-related semantics from a data definition language.

Byte: This document uses the term "byte" in its now-customary sense as a synonym for "octet".

1.2.2. Conceptual Terms

Thing: A physical item that is also available for interaction over a network.

Element: A part or an aspect of something abstract; i.e., the term is used here in its usual English definition.

Affordance: An element of an interface offered for interaction. Such an element becomes an Affordance when information is available (directly or indirectly) that indicates how it can or should be used. In the present document, the term is used for the digital (network-directed) interfaces of a Thing only; as it is a physical object as well, the Thing might also have physical affordances such as buttons, dials, and displays. The specification language offers certain ways to create sets of related Affordances and combine them into "Groupings" (see below).

Property: An Affordance that can potentially be used to read, write, and/or observe state (current/stored information) on a Grouping.

Action: An Affordance that can potentially be used to perform a named operation on a Grouping.

Event: An Affordance that can potentially be used to obtain information about what happened to a Grouping.

1.2.3. Specification Language Terms

SDF Document: Container for SDF Definitions, together with data about the SDF Document itself (information block). Represented as a JSON text representing a single JSON map, which is built from nested maps.

SDF Model: Definitions and declarations that model the digital interaction opportunities offered by one or more kinds of Things, represented by Groupings (sdfObjects and sdfThings). An SDF Model can be fully contained in a single SDF Document, or it can be built from an SDF Document that references definitions and declarations from additional SDF documents. The term SDF Specification can be used when the distinction between the distribution into individual SDF Documents and the abstract nature of the SDF Model is not of interest.

Block: One or more entries in a JSON map that is part of an SDF specification. These entries can be described as a Block to emphasize that they serve a specific function together.

Group: An entry in the top-level JSON map that represents the SDF document. Groups also can be used in certain nested definitions. A group has a Class Name Keyword as its key and a map of named definition entries (Definition Group) as a value.

Class Name Keyword: One of `sdfThing`, `sdfObject`, `sdfProperty`, `sdfAction`, `sdfEvent`, or `sdfData`. The Classes for these type keywords are capitalized and prefixed with `sdf`.

Class: Abstract term for the information that is contained in groups identified by a Class Name Keyword.

Quality: A metadata item in a definition or declaration that says something about that definition or declaration. A quality is represented in SDF as an entry in a JSON map (JSON object) that serves as a definition or declaration. (The term "Quality" is used because another popular term, "Property", already has a different meaning.)

Definition: An entry in a Definition Group. The entry creates a new semantic term for use in SDF models and associates it with a set of qualities. Unless the Class Name Keyword of the Group also makes it a Declaration (see Section 3.3), a definition just defines a term and it does not create a component item within the enclosing definition.

Declaration: A definition within an enclosing definition that is intended to create a component item within that enclosing definition. Every declaration can also be used as a definition for reference elsewhere.

Grouping: An sdfThing or sdfObject, i.e., (directly or indirectly) a description for a combination of Affordances.

Object, sdfObject: A Grouping where the declarations that it contains are for Affordances only (Property, Action, and Event declarations). It serves as the main "atom" of reusable semantics for model construction, representing the interaction model for a Thing that is simple enough to not require a nested structure. Therefore, sdfObjects are similar to sdfThings, but do not allow nesting, i.e., they cannot contain other Groupings (sdfObjects or sdfThings).

sdfThing: A Grouping that can contain nested Groupings (sdfThings and sdfObjects). Like sdfObject, it can also contain Affordance declarations (Property, Action, and Event declarations). (Note that "Thing" has a different meaning from sdfThing and is therefore not available as a colloquial shorthand of sdfThing.)

Augmentation Mechanism: A companion document to a base SDF Model that provides additional information ("augments" the base specification). The information may be for use in a specific ecosystem or with a specific protocol ("Protocol Binding"). No specific Augmentation Mechanisms are defined in base SDF. A simple mechanism for such augmentations has been discussed as a "mapping file" [SDF-MAPPING].

Protocol Binding: A companion document to an SDF Model that defines how to map the abstract concepts in the model into the protocols that are in use in a specific ecosystem. The Protocol Binding might supply URL components, numeric IDs, and similar details. Protocol Bindings are one case of an Augmentation Mechanism.

1.2.4. Conventions

Regular expressions that are used in the text as a "pattern" for some string are interpreted as per [RFC9485]. (Note that a form of regular expressions is also used as values of the quality `pattern`; see Appendix C.2.)

The term "URI" in this document always refers to "full" URIs ("`URI`" in Section 3 of RFC 3986 [STD66]), never to relative URI references ("`relative-ref`" in Section 4.1 of RFC 3986 [STD66]), so the term "URI" does *NOT* serve as the colloquial abbreviation of "URI-Reference" it is often used for. Therefore, the "reference resolution" process defined in Section 5 of RFC 3986 [STD66] is *NOT* used in this specification. Where necessary, full URIs are assembled out of substrings by simple concatenation, e.g., when CURIEs are expanded (Section 4.3) or when a global name is formed out of a namespace `absolute-URI` (Section 5 of RFC 3986 [STD66]) and a fragment identifier part (Section 4.1). Also note that URIs are not only used to construct the SDF models, they are also the *subject* of SDF models where they are used as data in actual interactions (and could even be represented as relative references there); these two usages are entirely separate.

The singular form is chosen as the preferred one for the keywords defined in this specification.

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Overview

2.1. Example Definition

The overview starts with an example for the SDF definition of a simple sdfObject called "Switch" (Figure 1).

```
{
  "info": {
    "title": "Example document for SDF (Semantic Definition Format)",
    "version": "2019-04-24",
    "copyright": "Copyright 2019 Example Corp. All rights reserved.",
    "license": "https://example.com/license"
  },
  "namespace": {
    "cap": "https://example.com/capability/cap"
  },
  "defaultNamespace": "cap",
  "sdfObject": {
    "Switch": {
      "sdfProperty": {
        "value": {
          "description":
"The state of the switch; false for off and true for on.",
          "type": "boolean"
        }
      },
      "sdfAction": {
        "on": {
          "description":
"Turn the switch on; equivalent to setting value to true."
        },
        "off": {
          "description":
"Turn the switch off; equivalent to setting value to false."
        },
        "toggle": {
          "description":
"Toggle the switch; equivalent to setting value to its complement."
        }
      }
    }
  }
}
```

Figure 1: A Simple Example of an SDF Document

This is a model of a switch. The state `value` declared in the `sdfProperty` group, represented by a Boolean, will be true for "on" and will be false for "off". The Actions `on` or `off` declared in the `sdfAction` group are redundant with setting the `value` and are in the example to illustrate that there are often different ways of achieving the same effect. The action `toggle` will invert the value of the sdfProperty value so that 2-way switches can be created; having such action will avoid the need for retrieving the current value first and then applying/setting the inverted value.

The `sdfObject` group lists the affordances of Things modeled by this sdfObject. The `sdfProperty` group lists the Property affordances described by the model; these represent various perspectives on the state of the sdfObject. Properties can have additional qualities to describe the state more precisely. Properties can be annotated to be read, write, or read/write; how this is actually done by the underlying transfer protocols is not described in the SDF model but left to companion protocol bindings. Properties are often used with RESTful paradigms [REST-IOT]

describing state. The `sdfAction` group is the mechanism to describe other interactions in terms of their names, input, and output data (no data are used in the example), as in a POST method in REST or in a remote procedure call. The example `toggle` is an Action that changes the state based on the current state of the Property named `value`. (The third type of affordance is Events, which are not described in this example.)

In the JSON representation, the `info` group is an exception in that this group's map has keys taken from the SDF vocabulary. All other groups (such as `namespace` and `sdfObject`) have maps with keys that are freely defined by the model writer (`Switch`, `value`, `on`, etc.). These map keys are therefore called *Given Names*. The groups made up of entries with Given Names as keys usually use the `named<>` production in the formal syntax of SDF (Appendix A). Where the values of these entries are maps, these again use SDF vocabulary keys, and so on, generally alternating in further nesting. The SDF-defined vocabulary items used in the hierarchy of such groups are often, but not always, called *Quality Names* or *qualities*. See Section 2.3 for more information about naming in SDF.

2.2. Elements of an SDF Model

The SDF language uses six predefined Class Name Keywords for modeling connected Things, which are illustrated in Figure 2 (limited rendition in the plaintext form of this document, please use typographic forms for full information).

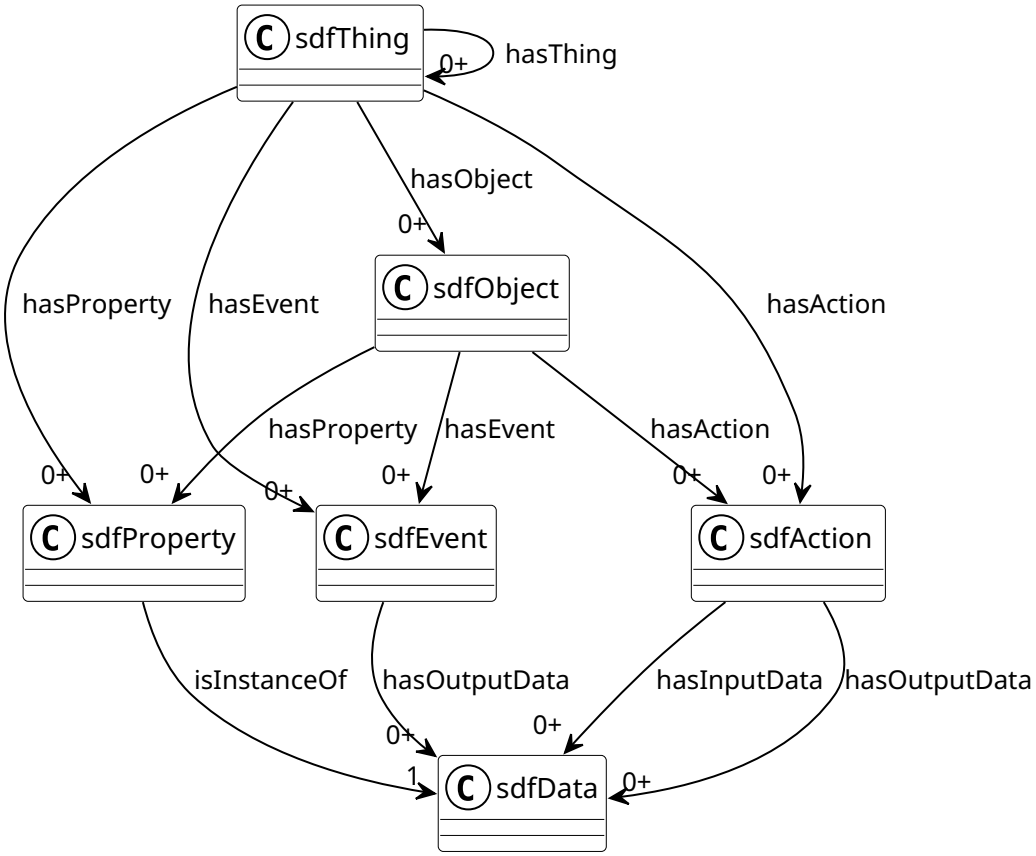

Figure 2: Main Classes Used in SDF Models

The six main Class Name Keywords are discussed below.

2.2.1. sdfObject

sdfObjects, the items listed in an `sdfObject` definition group, are the main "atom" of reusable semantics for model construction. The concept aligns in scope with common definition items from many IoT modeling systems, e.g., ZigBee Clusters [ZCL], OMA SpecWorks LwM2M Objects [OMA], OCF Resource Types [OCF], and W3C Web of Things Thing Descriptions [WoT].

An sdfObject definition contains a set of `sdfProperty`, `sdfAction`, and `sdfEvent` definitions that describe the interaction affordances associated with some scope of functionality.

For the granularity of definition, sdfObject definitions are meant to be kept narrow enough in scope to enable broad reuse and interoperability. For example, defining a light bulb using separate sdfObject definitions for on/off control, dimming, and color control affordances will enable interoperable functionality to be configured for diverse product types. An sdfObject definition for a common on/off control may be used to control many different kinds of Things that require on/off control.

The presence of one or both of the optional qualities "`minItems`" and "`maxItems`" defines the sdfObject as an array, i.e., all the affordances defined for the sdfObject exist a number of times, indexed by a number constrained to be between `minItems` and `maxItems`, inclusive, if given. (Note: Setting "`minItems`" to zero and leaving out "`maxItems`" puts the minimum constraints on that array.)

2.2.2. sdfProperty

`sdfProperty` is used to model elements of state within Things modeled by the enclosing grouping.

A named definition entry in an sdfProperty may be associated with some protocol affordance to enable the application to obtain the state variable and, optionally, modify the state variable. Additionally, some protocols provide for in-time reporting of state changes. (These three aspects are described by the qualities `readable`, `writable`, and `observable` defined for an sdfProperty.)

Definitions in `sdfProperty` groups look like the definitions in `sdfData` groups. However, they actually declare that a Property with the given qualities potentially is present in the containing sdfObject.

For definitions in `sdfProperty` and `sdfData`, SDF provides qualities that can constrain the structure and values of data allowed in the interactions modeled by them. It also provides qualities that associate semantics to this data, such as engineering units and unit scaling information.

For the data definition within `sdfProperty` or `sdfData`, SDF borrows some vocabulary proposed for drafts 4 [JSO4] [JSO4V] and 7 [JSO7] [JSO7V] of the json-schema.org "JSON Schema" format (collectively called JSO here), enhanced by qualities that are specific to SDF. Details about the JSO-inspired vocabulary are in Appendix C. For base SDF, data are constrained to be of simple types (number, string, boolean), JSON maps composed of named data, and arrays of these types. Syntax extension points are provided that can be used to provide richer types in a future extension of this specification (possibly more of which can be borrowed from json-schema.org).

Note that sdfProperty definitions (and sdfData definitions in general) are not intended to constrain the formats of data used for communication over network interfaces. Where needed, data definitions for payloads of protocol messages are expected to be part of the protocol binding.

2.2.3. sdfAction

The `sdfAction` group contains declarations of Actions, which model affordances that, when triggered, have an effect that can go beyond just reading, updating, or observing Thing state. Actions often result in some outward physical effect (which, itself, cannot be modeled in SDF). From a programmer's perspective, they might be considered to be roughly analogous to method calls.

Actions may have data parameters; these are each modeled as a single item of input data and output data. Where multiple parameters need to be modeled, an "`object`" type can be used to combine these parameters into one; for an example, see Figure 6 in Appendix C.5.

Actions may be long-running, that is to say that the effects may not take place immediately as would be expected for an update to an sdfProperty; the effects may play out over time and emit action results. Actions may also not always complete and may result in application errors, such as an item blocking the closing of an automatic door.

One idiom for giving an action initiator status and control about the ongoing action is to provide a URI for an ephemeral "action resource" in the sdfAction output data, allowing the action to deliver immediate feedback (including errors that prevent the action from starting) and the action initiator to use the action resource for further observation or modification of the ongoing action (including canceling it). Base SDF does not provide any tailored support for describing such action resources; an extension for modeling links in more detail (for instance, [SDFTYPE-LINK]) may be all that is needed to fully enable modeling them.

Actions may have (or lack) the characteristics of idempotence and side-effect safety (see Section 9.2 of RFC 9110 [STD97] for more on these terms).

Base SDF only provides data constraint modeling and semantics for the input and output data of definitions in `sdfAction` groups. Again, data definitions for payloads of protocol messages, and detailed protocol settings for invoking the action, are expected to be part of the protocol binding.

2.2.4. sdfEvent

The `sdfEvent` group contains declarations of Events, which model affordances that inform about "happenings" associated with a Thing modeled by the enclosing sdfObject; these may result in a signal being stored or emitted as a result.

Note that there is a trivial overlap with sdfProperty state changes, which may also be defined as Events but are not generally required to be defined as such. However, Events may exhibit certain ordering, consistency, and reliability requirements that are expected to be supported in various implementations of sdfEvent that do distinguish sdfEvent from sdfProperty. For instance, while a state change may simply be superseded by another state change, some Events are "precious" and need to be preserved even if further Events follow.

Base SDF only provides data constraint modeling and semantics for the output data of Event affordances. Again, data definitions for payloads of protocol messages, and detailed protocol settings for soliciting the event, are expected to be part of the protocol binding.

2.2.5. sdfData

Definitions in `sdfData` groups do not themselves specify affordances. These definitions are provided separately from those in sdfProperty groups to enable common modeling patterns, data constraints, and semantic anchor concepts to be factored out for data items that make up sdfProperty items and serve as input and output data for sdfAction and sdfEvent items. The data types defined in sdfData definitions only spring to life by being referenced in one of these contexts (directly or indirectly via some other sdfData definitions).

It is a common use case for such a data definition to be shared between an sdfProperty item and input or output parameters of an sdfAction or output data provided by an sdfEvent. sdfData definitions also enable factoring out extended application data types, such as mode and machine state enumerations to be reused across multiple definitions that have similar basic characteristics and requirements.

2.2.6. sdfThing

Back at the top level, the `sdfThing` group enables definition of models for complex devices that will use one or more sdfObject definitions. Like sdfObject, sdfThing groups allow for the inclusion of interaction affordances, sdfData, as well as "`minItems`" and "`maxItems`" qualities. Therefore, they can be seen as a superset of sdfObject groups, additionally allowing for composition.

As a result, an sdfThing directly or indirectly contains a set of sdfProperty, sdfAction, and sdfEvent definitions that describe the interaction affordances associated with some scope of functionality.

A definition in an sdfThing group can refine the metadata of the definitions it is composed of: other definitions in sdfThing groups or definitions in sdfObject groups.

2.3. Member Names: Given Names and Quality Names

SDF documents are JSON maps that mostly employ JSON maps as member values, which in turn mostly employ JSON maps as their member values, and so on. This nested structure of JSON maps creates a tree, where the edges are the member names (map keys) used in these JSON maps. (In certain cases, where member names are not needed, JSON arrays may be interspersed in this tree.)

2.3.1. Given Names and Quality Names

For any particular JSON map in an SDF document, the set of member names that are used is either:

- A set of "*Quality Names*", where the entries in the map are Qualities. Quality Names are defined by the present specification and its extensions, together with specific semantics to be associated with the member value given with a certain Quality Name.
- A set of "*Given Names*", where the entries in the map are separate entities (definitions, declarations, etc.) that each have names that are chosen by the SDF document author in order that these names can be employed by a user of that model.

In a path from the root of the tree to any leaf, Quality Names and Given Names roughly alternate (with the information block, Section 3.1, as a prominent exception).

The meaning of the JSON map that is the member value associated with a Given Name is derived from the Quality Name that was used as the member name associated to the parent. In the CDDL grammar given in Appendix A, JSON maps with member names that are Given Names are defined using the CDDL generic rule reference `named<membervalues>`, where `membervalues` is in

turn the structure of the member values of the JSON map, i.e., the value of the member named by the Given Name. As quality-named maps and given-named maps roughly alternate in a path down the tree, `membervalues` is usually a map built from Quality Names as keys.

2.3.2. Hierarchical Names

From the outside of a specification, Given Names are usually used as part of a hierarchical name that looks like a JSON Pointer [RFC6901]. These hierarchical names are generally rooted in (used as the fragment identifier in) an outer namespace that looks like an `https://` URL (see Section 4).

As Quality Names and Given Names roughly alternate in a path into the model, the JSON Pointer part of the hierarchical name also alternates between Quality Names and Given Names.

Note that the actual Given Names may need to be encoded when specified via the JSON Pointer fragment identifier syntax. There are two layers of such encoding: tilde encoding of `~` and `/` as per Section 3 of [RFC6901], as well as percent encoding of the tilde-encoded name into a valid URI fragment as per Section 6 of [RFC6901]. For example, when a model is using the Given Name

```
warning/danger alarm
```

(with an embedded slash and a space) for an sdfObject, that sdfObject may need to be referenced as

```
#/sdfObject/warning~1danger%20alarm
```

To sidestep potential interoperability problems, it is probably wise to avoid characters in Given Names that need such encoding (Quality Names are already defined in such a way that they never do).

2.3.3. Extensibility of Given Names and Quality Names

In SDF, both Quality Names and Given Names are *extension points*. This is more obvious for Quality Names. Extending SDF is mostly done by defining additional qualities. To enable non-conflicting third party extensions to SDF, qualified names (names with an embedded colon) can be used as Quality Names.

A nonqualified Quality Name is composed of ASCII letters, digits, and $ signs, starting with a lowercase letter or a $ sign (i.e., using a pattern of "`[a-z$][A-Za-z$0-9]*`"). Names with $ signs are intended to be used for functions separate from most other names; for instance, `$comment` is used for the comment quality in this specification (the presence or absence of a `$comment` quality does not change the meaning of the SDF model). Names that are composed of multiple English words can use the "lowerCamelCase" convention [CamelCase] for indicating the word boundaries; no other use is intended for upper case letters in Quality Names.

A qualified Quality Name is composed of a Quality Name Prefix, a `:` (colon) character, and a nonqualified Quality Name. Quality Name Prefixes are registered in the "Quality Name Prefixes" registry in the "Semantic Definition Format (SDF)" registry group (Section 7.5.2). They are composed of lowercase ASCII letters and digits, starting with a lowercase ASCII letter (i.e., using a pattern of "`[a-z][a-z0-9]*`").

Given Names are not restricted by the formal SDF syntax. To enable non-surprising name translations in tools, combinations of ASCII alphanumeric characters and `-` (ASCII hyphen/minus) are preferred, typically employing kebab-case for names constructed out of multiple words [KebabCase]. ASCII hyphen/minus can then unambiguously be translated to an ASCII `_` underscore character and back depending on the programming environment. Some styles also allow a dot ("`.`") in Given Names. Given Names are often sufficiently self-explanatory that they can be used in place of the `label` quality if that is not given. In turn, if a Given Name turns out too complicated, a more elaborate `label` can be given and the Given Name kept simple. As Given Names are "programmers' names", base SDF does not address internationalization of Given Names. (More likely qualities to receive localizable equivalents by exercising the Quality Name extension point are `label` and `description`).

Further, to enable Given Names to have a more powerful role in building global hierarchical names, an extension is foreseen that makes use of qualified names for Given Names. So, until that extension is defined, Given Names with one or more embedded colons are reserved and **MUST NOT** be used in an SDF document.

All names in SDF are case-sensitive.

3. SDF Structure

SDF definitions are contained in SDF documents together with data about the SDF document itself (information block). Definitions and declarations from additional SDF documents can be referenced; together with the definitions and declarations in the referencing SDF document, they build the SDF model expressed by that SDF document.

Each SDF document is represented as a single JSON map. This map can be thought of as having three blocks: the information block, the namespaces block, and the definitions block. These blocks contain zero or more JSON name/value pairs, the names of which are Quality Names and the values of which mostly are (nested) maps (the exception defined in base SDF is the defaultNamespace quality, the value of which is a text string). An empty nested map of this kind is equivalent to not having the quality included at all.

3.1. Information Block

The information block contains generic metadata for the SDF document itself and all included definitions. To enable tool integration, the information block is optional in the grammar of SDF; most processes for working with SDF documents will have policies that only SDF documents with an info block can be processed. It is therefore **RECOMMENDED** that SDF validator tools emit a warning when no information block is found.

The keyword (map key) that defines an information block is "info". The keyword's value is a nested JSON map with a set of entries that represent qualities that apply to the included definitions.

Qualities of this map are shown in Table 1. None of these qualities are required or have default values that are assumed if the quality is absent.

Quality	Type	Description
title	string	A short summary to be displayed in search results, etc.
description	string	Long-form text description (no constraints)
version	string	The incremental version of the definition
modified	string	Time of the latest modification
copyright	string	Link to text or embedded text containing a copyright notice
license	string	Link to text or embedded text containing license terms
features	array of strings	List of extension features used
$comment	string	Source code comments only, no semantics

Table 1: Qualities of the Information Block

The version quality is used to indicate version information about the set of definitions in the SDF document. The version is **RECOMMENDED** to be lexicographically increasing over the life of a model; a newer model always has a version string that string-compares higher than all previous versions. This is easily achieved by following the convention to start the version with a `date-time` as defined in [RFC3339] or, if new versions are generated less frequently than once a day, just the `full-date` (i.e., YYYY-MM-DD); in many cases, that will be all that is needed (see Figure 1 for an example). This specification does not give a strict definition for the format of the version string, but each system or organization using the version string should define internal structure and semantics to the level needed for their use. If no further details are provided, a `date-time` or `full-date` in this field can be assumed to indicate the latest update time of the definitions in the SDF document.

The modified quality can be used with a value using `date-time` as defined in [RFC3339] (with `Z` for time-zone) or `full-date` format to express time of the latest revision of the definitions.

The license string is preferably either a URI that points to a web page with an unambiguous definition of the license or an [SPDX] license identifier. (As an example, for models to be handled by the One Data Model liaison group, this license identifier will typically be "BSD-3-Clause".)

The `features` quality can be used to list names of critical (i.e., cannot be safely ignored) SDF extension features that need to be understood for the definitions to be properly processed. Extension feature names will be specified in extension documents. They can either be registered

(see Section 7.5.4 for specifics, which make sure that a registered feature name does not contain a colon) or be a URI (which always contain a colon). Note that SDF processors are not expected to, and normally **SHOULD NOT**, dereference URIs used as feature names; any representation retrievable under such a URI could be useful to humans, though. (See [DEREF-ID-PATTERN] for a more extensive discussion of dereferenceable identifiers).

3.2. Namespaces Block

The namespaces block contains the `namespace` map and the `defaultNamespace` setting; none of these qualities are required or have default values that are assumed if the quality is absent.

The namespace map is a map from short names for URIs to the namespace URIs themselves.

The defaultNamespace setting selects one of the entries in the namespace map by giving its short name. The associated URI (value of this entry) becomes the default namespace for the SDF document.

Quality	Type	Description
namespace	map	Defines short names mapped to namespace URIs, to be used as identifier prefixes
defaultNamespace	string	Identifies one of the prefixes in the namespace map to be used as a default in resolving identifiers

Table 2: Namespaces Block

The following example declares a set of namespaces and defines `cap` as the default namespace. By convention, the values in the namespace map contain full URIs without a fragment identifier and the fragment identifier is then added, if needed, where the namespace entry is used.

```
"namespace": {
  "cap": "https://example.com/capability/cap",
  "zcl": "https://zcl.example.com/sdf"
},
"defaultNamespace": "cap"
```

Multiple SDF documents can contribute to the same namespace by using the same namespace URI for the default namespace across the documents.

If no defaultNamespace setting is given, the SDF document does not contribute to a global namespace (all definitions remain local to the model and are not accessible for re-use by other models). As the defaultNamespace is set by supplying a namespace short name, its presence requires a namespace map that contains a mapping for that namespace short name.

If no namespace map is given, no short names for namespace URIs are set up and no defaultNamespace can be given.

3.3. Definitions Block

The Definitions block contains one or more groups, each identified by a Class Name Keyword such as `sdfObject` or `sdfProperty`. There can only be one group per keyword at this level; putting all the individual definitions in the group under that keyword is just a shortcut for identifying the class name keyword that applies to each of them without repeating it for each definition.

The value of each group is a JSON map, the keys of which serve for naming the individual definitions in this group, and the corresponding values provide a set of qualities (name-value pairs) for the individual definition. (In short, these map entries are also termed "named sets of qualities".)

Each group may contain zero or more definitions. Each identifier defined creates a new type and term in the target namespace. Declarations have a scope of the definition block they are directly contained in.

In turn, a definition may contain other definitions. Each definition is a named set of qualities, i.e., it consists of the newly defined identifier and a set of key-value pairs that represent the defined qualities and contained definitions.

An example for an sdfObject definition is given in Figure 3:

```
"sdfObject": {
  "foo": {
    "sdfProperty": {
      "bar": {
        "type": "boolean"
      }
    }
  }
}
```

Figure 3: Example sdfObject Definition

This example defines an sdfObject "foo" that is defined in the default namespace (full address: `#/sdfObject/foo`), containing a Property that can be addressed as `#/sdfObject/foo/sdfProperty/bar`, with data of type boolean.

Often, definitions are also declarations. The definition of the entry "bar" in the Property "foo" means that data corresponding to the "foo" Property in a Property interaction offered by Thing can have zero or one components modeled by "bar". Entries within `sdfProperty`, `sdfAction`, and `sdfEvent` that are in turn within `sdfObject` or `sdfThing` entries, are also declarations; entries within `sdfData` are not. Similarly, `sdfObject` or `sdfThing` entries within an sdfThing definition specify that the interactions offered by a Thing modeled by this sdfThing include the interactions modeled by the nested `sdfObject` or `sdfThing`.

3.4. Top-Level Affordances and sdfData

Besides their placement within an sdfObject or sdfThing, affordances (i.e., `sdfProperty`, `sdfAction`, and `sdfEvent`) as well as `sdfData` can also be placed at the top level of an SDF document. Since they are not associated with an sdfObject or sdfThing, these kinds of definitions are intended to be reused via the `sdfRef` mechanism (see Section 4.4).

4. Names and Namespaces

SDF documents may contribute to a global namespace and may reference elements from that global namespace. (An SDF document that does not set a defaultNamespace does not contribute to a global namespace.)

4.1. Structure

Global names look exactly like `https://` URIs with attached fragment identifiers.

There is no intention to require that these URIs can be dereferenced. (However, as future extensions of SDF might find a use for dereferencing global names, the URI should be chosen in such a way that this may become possible in the future. See also [DEREF-ID-PATTERN] for a discussion of dereferenceable identifiers.)

The absolute-URI of a global name should be a URI as per Section 3 of RFC 3986 [STD66] with a scheme of "https" and a path (`hier-part` in [STD66]). For base SDF, the query part should not be used (it might be used in extensions).

The fragment identifier is constructed as per Section 6 of [RFC6901].

4.2. Contributing Global Names

The fragment identifier part of a global name defined in an SDF document is constructed from a JSON Pointer that selects the element defined for this name in the SDF document. The absolute-URI part is a copy of the default namespace.

As a result, the default namespace is always the target namespace for a name for which a definition is contributed. In order to emphasize that name definitions are contributed to the default namespace, this namespace is also termed the "target namespace" of the SDF document.

For instance, in Figure 1, definitions for the following global names are contributed:

- https://example.com/capability/cap#/sdfObject/Switch
- https://example.com/capability/cap#/sdfObject/Switch/sdfProperty/value
- https://example.com/capability/cap#/sdfObject/Switch/sdfAction/on
- https://example.com/capability/cap#/sdfObject/Switch/sdfAction/off
- https://example.com/capability/cap#/sdfObject/Switch/sdfAction/toggle

Note the #, which separates the absolute-URI part (Section 4.3 of RFC 3986 [STD66]) from the fragment identifier part (including the #, a JSON Pointer as in Section 6 of [RFC6901]).

4.3. Referencing Global Names

A name reference takes the form of the production `curie` in Section 3 of [W3C.NOTE-curie-20101216], but limiting the IRIs involved in that grammar to URIs as per [STD66] and the prefixes to ASCII characters [STD80]. (Note that this definition does not make use of the production `safe-curie` in [W3C.NOTE-curie-20101216].)

A name that is contributed by the current SDF document can be referenced by a Same-Document Reference as per Section 4.4 of RFC 3986 [STD66]. As there is little point in referencing the entire SDF document, this will be a # followed by a JSON Pointer. This is the only kind of name reference to itself that is possible in an SDF document that does not set a default namespace.

Name references that point outside the current SDF document need to contain CURIE prefixes. These then reference namespace declarations in the namespaces block.

For example, if a namespace prefix is defined:

```
"namespace": {
  "foo": "https://example.com/"
}
```

then this reference to that namespace:

```
"sdfRef": "foo:#/sdfData/temperatureData"
```

references the global name:

```
"https://example.com/#/sdfData/temperatureData"
```

Note that there is no way to provide a URI scheme name in a CURIE, so all references to outside of the document need to go through the namespace map.

Name references occur only in specific elements of the syntax of SDF:

- copying elements via sdfRef values
- pointing to elements via sdfRequired value elements

4.4. sdfRef

In a JSON map establishing a definition, the keyword `sdfRef` is used to copy the qualities and enclosed definitions of the referenced definition, indicated by the included name reference, into the newly formed definition. (This can be compared to the processing of the `$ref` keyword in

[JSO7].) The referenced definition should be such that, after copying and applying the additional qualities in the referencing definition, the newly built definition is also valid SDF (e.g., the copied qualities and definitions are valid in the context of the new definition).

For example, this reference:

```
"temperatureProperty": {
  "sdfRef": "#/sdfData/temperatureData"
}
```

creates a new definition "temperatureProperty" that contains all of the qualities defined in the definition at /sdfData/temperatureData.

The sdfRef member need not be the only member of a map. Additional members may be present with the intention of overriding parts of the referenced map or adding new qualities or definitions.

When processing sdfRef, if the target definition contains also sdfRef (i.e., is based on yet another definition), that **MUST** be processed as well.

More formally, for a JSON map that contains an sdfRef member, the semantics are defined to be as if the following steps were performed:

1. The JSON map that contains the sdfRef member is copied into a variable named "patch".
2. The sdfRef member of the copy in "patch" is removed.
3. The JSON Pointer that is the value of the sdfRef member is dereferenced and the result is copied into a variable named "original".
4. The JSON Merge Patch algorithm [RFC7396] is applied to patch the contents of "original" with the contents of "patch".
5. The result of the Merge Patch is used in place of the value of the original JSON map.

Note that the formal syntaxes given in Appendices A and B generally describe the *result* of applying a merge-patch. The notations are not powerful enough to describe, for instance, how the merge-patch algorithm causes null values within the sdfRef to remove members of JSON maps from the referenced target. Nonetheless, the syntaxes also give the syntax of the sdfRef itself, which vanishes during the resolution; therefore, in many cases, even merge-patch inputs will validate with these formal syntaxes.

Given the example (Figure 1) and the following definition:

```
{
  "info": {
    "title": "Example light switch using sdfRef"
  },
  "namespace": {
    "cap": "https://example.com/capability/cap"
  },
  "defaultNamespace": "cap",
  "sdfObject": {
    "BasicSwitch": {
      "sdfRef": "cap:#/sdfObject/Switch",
      "sdfAction": {
        "toggle": null
      }
    }
  }
}
```

The resulting definition of the "BasicSwitch" sdfObject would be identical to the definition of the "Switch" sdfObject, except it would not contain the "toggle" Action.

```
{
  "info": {
    "title": "Example light switch using sdfRef"
  },
  "namespace": {
    "cap": "https://example.com/capability/cap"
  },
  "defaultNamespace": "cap",
  "sdfObject": {
    "BasicSwitch": {
      "sdfProperty": {
        "value": {
          "description":
"The state of the switch; false for off and true for on.",
          "type": "boolean"
        }
      },
      "sdfAction": {
        "on": {
          "description":
"Turn the switch on; equivalent to setting value to true."
        },
        "off": {
          "description":
"Turn the switch off; equivalent to setting value to false."
        }
      }
    }
  }
}
```

4.4.1. Resolved Models

A model where all sdfRef references are processed as described in Section 4.4 is called a resolved model.

For example, given the following sdfData definitions:

```
"sdfData": {
  "Coordinate" : {
    "type": "number", "unit": "m"
  },
  "X-Coordinate" : {
    "sdfRef" : "#/sdfData/Coordinate",
    "description":
"Distance from the base of the Thing along the X axis."
  },
  "Non-neg-X-Coordinate" : {
    "sdfRef": "#/sdfData/X-Coordinate",
    "minimum": 0
  }
}
```

the definitions would look as follows after being resolved:

```
"sdfData": {
  "Coordinate" : {
    "type": "number", "unit": "m"
  },
  "X-Coordinate" : {
    "description":
"Distance from the base of the Thing along the X axis.",
    "type": "number", "unit": "m"
  },
  "Non-neg-X-Coordinate" : {
    "description":
"Distance from the base of the Thing along the X axis.",
    "minimum": 0, "type": "number", "unit": "m"
  }
}
```

4.5. sdfRequired

The keyword `sdfRequired` is provided to apply a constraint that defines for which declarations the corresponding data are mandatory in a grouping (sdfThing or sdfObject) modeled by the current definition.

The value of `sdfRequired` is an array of references, each indicating one or more declarations that are mandatory to be represented.

References in this array can be SDF names (JSON Pointers) or one of two abbreviated reference formats:

- A text string with a "referenceable-name", namely an affordance name or a grouping name:
  - All affordance declarations that are directly in the same grouping (i.e., not nested further in another grouping) and that carry this name are declared to be mandatory to be represented. Note that there can be multiple such affordance declarations, one per affordance type.
  - The same applies to groupings made mandatory within groupings containing them.
- The Boolean value `true`. The affordance or grouping itself that carries the `sdfRequired` keyword is declared to be mandatory to be represented.

Note that referenceable-names are not subject to the encoding JSON Pointers require as discussed in Section 2.3.2. To ensure that referenceable-names are reliably distinguished from JSON Pointers, they are defined such that they cannot contain ":" or "#" characters (see rule `referenceable-name` in Appendix A). (If these characters are indeed contained in a Given Name, a JSON Pointer needs to be formed instead in order to reference it in "sdfRequired", potentially requiring further path elements as well as JSON Pointer encoding. The need for this is best avoided by choosing Given Names without these characters.)

The example in Figure 4 shows two required elements in the sdfObject definition for "temperatureWithAlarm", the sdfProperty "currentTemperature", and the sdfEvent "overTemperatureEvent". The example also shows the use of JSON Pointers with "sdfRef" to use a pre-existing definition for the sdfProperty "currentTemperature" and for the sdfOutputData produced by the sdfEvent "overTemperatureEvent".

```
"sdfObject": {
  "temperatureWithAlarm": {
    "sdfRequired": [
"#/sdfObject/temperatureWithAlarm/sdfProperty/currentTemperature",
"#/sdfObject/temperatureWithAlarm/sdfEvent/overTemperatureEvent"
    ],
    "sdfData":{
       "temperatureData": {
        "type": "number"
      }
    },
    "sdfProperty": {
      "currentTemperature": {
"sdfRef": "#/sdfObject/temperatureWithAlarm/sdfData/temperatureData",
      "writable": false
      }
    },
    "sdfEvent": {
      "overTemperatureEvent": {
       "sdfOutputData": {
"sdfRef": "#/sdfObject/temperatureWithAlarm/sdfData/temperatureData"
        }
      }
    }
  }
}
```

Figure 4: Using sdfRequired

In Figure 4, the same sdfRequired can also be represented in short form:

```
    "sdfRequired": ["currentTemperature", "overTemperatureEvent"]
```

Or, for instance, "overTemperatureEvent" could carry:

```
      "overTemperatureEvent": {
        "sdfRequired": [true],
        "...": "..."
      }
```

4.6. Common Qualities

Definitions in SDF share a number of qualities that provide metadata for them. These are listed in Table 3. None of these qualities are required or have default values that are assumed if the quality is absent. If a short textual description is required for an application and no label is given in the SDF model, applications could use the last part (the last `reference-token`, Section 3 of [RFC6901]) of the JSON Pointer to the definition in its place.

Quality	Type	Description
description	string	long text (no constraints)
label	string	short text (no constraints)
$comment	string	source code comments only, no semantics
sdfRef	sdf-pointer	(see Section 4.4)
sdfRequired	pointer-list	(see Section 4.5, used in affordances or groupings)

Table 3: Common Qualities

4.7. Data Qualities

Data qualities are used in sdfData and sdfProperty definitions, which are named sets of data qualities (abbreviated as `named-sdq`).

These qualities include the common qualities, JSO-inspired qualities (see below), and data qualities defined specifically for the present specification; the latter are shown in Table 4.

Appendix C lists data qualities inspired by the various proposals at json-schema.org; the intention is that these (information model-level) qualities are compatible with the (data model) semantics from the versions of the json-schema.org proposal they were imported from.

Quality	Type	Description	Default
(common)		Section 4.6	
unit	string	unit name (note 1)	N/A
nullable	boolean	indicates a null value is available for this type	true
contentFormat	string	content type (IANA media type string plus parameters), encoding (note 2)	N/A
sdfType	string (Section 4.7.1)	sdfType enumeration (extensible)	N/A
sdfChoice	named set of data qualities (Section 4.7.2)	named alternatives	N/A

Quality	Type	Description	Default
enum	array of strings	abbreviation for string-valued named alternatives	N/A

Table 4: SDF-Defined Qualities of sdfData and sdfProperty

1. The unit name **SHOULD** be as per the "SenML Units" registry or the "Secondary Units" registry in [IANA.senml] as specified by Sections 4.5.2 and 12.1 of [RFC8428] and Section 3 of [RFC8798], respectively.

   Exceptionally, if a registration in these registries cannot be obtained or would be inappropriate, the unit name can also be a URI that is pointing to a definition of the unit. Note that SDF processors are not expected to, and normally **SHOULD NOT**, dereference these URIs; the definition pointed to may be useful to humans, though. (See [DEREF-ID-PATTERN] for a more extensive discussion of dereferenceable identifiers).

   A URI unit name is distinguished from a registered unit name by the presence of a colon; therefore, any registered unit names that contain a colon (at the time of writing, none) cannot be directly used in SDF.

   For use by translators into ecosystems that require URIs for unit names, the URN sub-namespace "urn:ietf:params:unit" is provided (Section 7.3). URNs from this sub-namespace **MUST NOT** be used in a `unit` quality in favor of simply notating the unit name (such as `kg` instead of `urn:ietf:params:unit:kg`) except where the unit name contains a colon and can therefore not be directly used in SDF.

2. The `contentFormat` quality follows the Content-Format-Spec as defined in Section 6 of [RFC9193], allowing for expressing both numeric and string based Content-Formats.

4.7.1. sdfType

SDF defines a number of basic types beyond those provided by JSON or JSO. These types are identified by the `sdfType` quality, which is a text string from a set of type names defined by the "sdfType values" registry in the "Semantic Definition Format (SDF)" registry group (Section 7.5.3). The sdfType name is composed of lowercase ASCII letters, digits, and `-` (ASCII hyphen/minus) characters, starting with a lowercase ASCII letter (i.e., using a pattern of "`[a-z][-a-z0-9]*`") and typically employing kebab-case for names constructed out of multiple words [KebabCase].

To aid interworking with JSO implementations, it is **RECOMMENDED** that sdfType is always used in conjunction with the `type` quality inherited from [JSO7V] in such a way as to yield a common representation of the type's values in JSON.

Values for sdfType that are defined in this specification are shown in Table 5. This table also gives a description of the semantics of the sdfType, the conventional value for `type` to be used with the sdfType value, and a conventional JSON representation for values of the type. The `type` and the JSON representation are chosen to be consistent with each other; this **MUST** be true for additionally registered sdfType values as well.

Name	Description	type	JSON Representation	Reference
byte-string	A sequence of zero or more bytes	string	base64url without padding	Section 3.4.5.2 of RFC 8949 [STD94]
unix-time	A point in civil time (note 1)	number	POSIX time	Section 3.4.2 of RFC 8949 [STD94]

Table 5: Values Defined in Base SDF for the sdfType Quality

(1) Note that the definition of `unix-time` does not imply the capability to represent points in time that fall on leap seconds. More date/time-related sdfTypes are likely to be added in the sdfType value registry.

4.7.2. sdfChoice

Data can be a choice of named alternatives called `sdfChoice`. Each alternative is identified by a name (string, key in the outer JSON map used to represent the overall choice) and a set of dataqualities (each in an inner JSON map, the value used to represent the individual alternative in the outer JSON map). Dataqualities that are specified at the same level as the sdfChoice apply to all choices in the sdfChoice except those specific choices where the dataquality is overridden at the choice level.

sdfChoice merges the functions of two constructs found in [JSO7V]:

- `enum`

  What could be expressed as:

```
"enum": ["foo", "bar", "baz"]
```

  in JSO, is often best represented as:

```
"sdfChoice": {
  "foo": { "description": "This is a foonly"},
  "bar": { "description":
           "As defined in the second world congress"},
  "baz": { "description": "From bigzee foobaz"}
}
```

  This allows the placement of other dataqualities such as `description` in the example.

  If an enum needs to use a data type different from the text string, what would, for instance, have been:

```
"type": "number",
"enum": [1, 2, 3]
```

in JSO, is represented as:

```
"type": "number",
"sdfChoice": {
  "a-better-name-for-alternative-1": { "const": 1 },
  "alternative-2": { "const": 2 },
  "the-third-alternative": { "const": 3 }
}
```

where the string names obviously would be chosen in a way that is descriptive for what these numbers actually stand for; sdfChoice also makes it easy to add number ranges into the mix.

(Note that `const` can also be used for strings as in the previous example, for instance, if the actual string value is indeed a crucial element for the data model.)

- anyOf

  JSO provides a type union called `anyOf`, which provides a choice between anonymous alternatives.

  What could have been in JSO:

```
"anyOf": [
  {"type": "array", "minItems": 3, "maxItems": "3",
   "items": {"$ref": "#/sdfData/rgbVal"}},
  {"type": "array", "minItems": 4, "maxItems": "4",
   "items": {"$ref": "#/sdfData/cmykVal"}}
]
```

  can be more descriptively notated in SDF as:

```
"sdfChoice": {
  "rgb": {"type": "array", "minItems": 3, "maxItems": "3",
          "items": {"sdfRef": "#/sdfData/rgbVal"}},
  "cmyk": {"type": "array", "minItems": 4, "maxItems": "4",
           "items": {"sdfRef": "#/sdfData/cmykVal"}}
}
```

Note that there is no need in SDF for the type intersection construct `allOf` or the peculiar type-xor construct `oneOf` found in [JSO7V].

As a simplification for users of SDF models who are accustomed to the JSO enum keyword, this is retained, but limited to a choice of text string values, such that:

```
"enum": ["foo", "bar", "baz"]
```

is syntactic sugar for:

```
"sdfChoice": {
  "foo": { "const": "foo"},
  "bar": { "const": "bar"},
  "baz": { "const": "baz"}
}
```

In a single definition, the keyword `enum` cannot be used at the same time as the keyword `sdfChoice`, as the former is just syntactic sugar for the latter.

5. Keywords for Definition Groups

The following SDF keywords are used to create definition groups in the target namespace. All these definitions share some common qualities as discussed in Section 4.6.

5.1. sdfObject

The `sdfObject` keyword denotes a group of zero or more sdfObject definitions. sdfObject definitions may contain or include definitions of named Properties, Actions, and Events declared for the sdfObject, as well as named data types (sdfData group) to be used in this or other sdfObjects.

The qualities of an sdfObject include the common qualities; additional qualities are shown in Table 6. None of these qualities are required or have default values that are assumed if the quality is absent.

Quality	Type	Description
(common)		Section 4.6
sdfProperty	property	zero or more named property definitions for this sdfObject
sdfAction	action	zero or more named action definitions for this sdfObject
sdfEvent	event	zero or more named event definitions for this sdfObject
sdfData	named-sdq	zero or more named data type definitions that might be used in the above
minItems	number	(array) minimum number of multiplied affordances in array
maxItems	number	(array) maximum number of multiplied affordances in array

Table 6: Qualities of sdfObject

5.2. sdfProperty

The `sdfProperty` keyword denotes a group of zero or more Property definitions.

Properties are used to model elements of state.

The qualities of a Property definition include the data qualities (and thus the common qualities); see Section 4.7. Additional qualities are shown in Table 7.

Quality	Type	Description	Default
(data)		Section 4.7	
readable	boolean	Reads are allowed	true
writable	boolean	Writes are allowed	true
observable	boolean	Flag to indicate asynchronous notification is available	true

Table 7: Qualities of sdfProperty

5.3. sdfAction

The `sdfAction` keyword denotes a group of zero or more Action definitions.

Actions are used to model commands and methods that are invoked. Actions may have parameter data that is supplied upon invocation and output data that is provided as a direct result of the invocation of the action (note that "action objects" may also be created to furnish ongoing information during a long-running action; these would be pointed to by the output data).

The qualities of an Action definition include the common qualities. Additional qualities are shown in Table 8. None of these qualities are required or have default values that are assumed if the quality is absent.

Quality	Type	Description
(common)		Section 4.6
sdfInputData	map	data qualities of the input data for an Action
sdfOutputData	map	data qualities of the output data for an Action
sdfData	named-sdq	zero or more named data type definitions that might be used in the above

Table 8: Qualities of sdfAction

`sdfInputData` defines the input data of the action. `sdfOutputData` defines the output data of the action. As discussed in Section 2.2.3, a set of data qualities with type "`object`" can be used to substructure either data item, with optionality indicated by the data quality `required`.

5.4. sdfEvent

The `sdfEvent` keyword denotes zero or more Event definitions.

Events are used to model asynchronous occurrences that may be communicated proactively. Events have data elements that are communicated upon the occurrence of the event.

The qualities of sdfEvent include the common qualities. Additional qualities are shown in Table 9. None of these qualities are required or have default values that are assumed if the quality is absent.

Quality	Type	Description
(common)		Section 4.6
sdfOutputData	map	data qualities of the output data for an Event
sdfData	named-sdq	zero or more named data type definitions that might be used in the above

Table 9: Qualities of sdfEvent

`sdfOutputData` defines the output data of the action. As discussed in Section 2.2.4, a set of data qualities with type "`object`" can be used to substructure the output data item, with optionality indicated by the data quality `required`.

5.5. sdfData

The `sdfData` keyword denotes a group of zero or more named data type definitions (named-sdq).

An sdfData definition provides a reusable semantic identifier for a type of data item and describes the constraints on the defined type. sdfData is not itself a declaration, so it does not cause any of these data items to be included in an affordance definition.

The qualities of sdfData include the data qualities (and thus the common qualities); see Section 4.7.

6. High-Level Composition

The requirements for high-level composition include the following:

- The ability to represent products, standardized product types, and modular products while maintaining the atomicity of sdfObjects.
- The ability to compose a reusable definition block from sdfObjects. Example: a single plug unit of an outlet strip with sdfObjects for on/off control, energy monitor, and optional dimmer, while retaining the atomicity of the individual sdfObjects.
- The ability to compose sdfObjects and other definition blocks into a higher level sdfThing that represents a product, while retaining the atomicity of sdfObjects.
- The ability to enrich and refine a base definition to have product-specific qualities and quality values, such as unit, range, and scale settings.

- The ability to reference items in one part of a complex definition from another part of the same definition. Example: summarizing the energy readings from all plugs in an outlet strip.

6.1. Paths in the Model Namespaces

The model namespace is organized according to terms that are defined in the SDF documents that contribute to the namespace. For example, definitions that originate from an organization or vendor are expected to be in a namespace that is specific to that organization or vendor.

The structure of a path in a namespace is defined by the JSON Pointers to the definitions in the SDF documents in that namespace. For example, if there is an SDF document defining an sdfObject "`Switch`" with an action "`on`", then the reference to the action would be "`ns:#/sdfObject/Switch/sdfAction/on`", where `ns` is the namespace prefix (short name for the namespace).

6.2. Modular Composition

Modular composition of definitions enables an existing definition (which could be in the same or another SDF document) to become part of a new definition by including a reference to the existing definition within the model namespace.

6.2.1. Use of the "sdfRef" Keyword to Reuse a Definition

An existing definition may be used as a template for a new definition, that is, a new definition is created in the target namespace that uses the defined qualities of some existing definition. This pattern uses the keyword `sdfRef` as a quality of a new definition with a value consisting of a reference to the existing definition that is to be used as a template.

In the definition that uses `sdfRef`, new qualities may be added and existing qualities from the referenced definition may be overridden. (Note that JSON maps do not have a defined order, so the SDF processor may see these overrides before seeing the `sdfRef`.)

Note that the definition referenced by `sdfRef` might contain qualities or definitions that are not valid in the context where the `sdfRef` is used. In this case, the resulting model, when resolved, may be invalid. Example: an sdfRef adds an sdfThing definition in an sdfObject definition.

As a convention, overrides are intended to be used only for further restricting the allowable set of data values. Such a usage is shown in Figure 5: any value allowable for a `cable-length` is also an allowable value for a `length`, with the additional restriction that the length cannot be smaller than 5 cm. (This is labeled as a convention as it cannot be checked in the general case. A quality of implementation consideration for a tool might be to provide at least some form of checking.) Note that the example provides a `description` that overrides the `description` of the referenced definition; as this quality is intended for human consumption, there is no conflict with the intended goal.

```
"sdfData":
  "length" : {
    "type": "number",
    "minimum": 0,
    "unit": "m"
    "description": "There can be no negative lengths."
  }
...
  "cable-length" : {
    "sdfRef": "#/sdfData/length"
    "minimum": 5e-2,
    "description": "Cables must be at least 5 cm."
  }
```

Figure 5: Using an Override to Further Restrict the Set of Data Values

6.3. sdfThing

An sdfThing is a set of declarations and qualities that may be part of a more complex model. For example, the sdfObject declarations that make up the definition of a single socket of an outlet strip could be encapsulated in an sdfThing, which itself could be used in a declaration in the sdfThing definition for the outlet strip. (See Figure 7 in Appendix D.1 for parts of an SDF model for this example.)

sdfThing definitions carry semantic meaning, such as a defined refrigerator compartment and a defined freezer compartment, making up a combination refrigerator-freezer product. An sdfThing may be composed of sdfObjects and other sdfThings. It can also contain sdfData definitions, as well as declarations of interaction affordances itself, such as a status (on/off) for the refrigerator-freezer as a whole (see Figure 8 in Appendix D.2 for an example SDF model illustrating these aspects).

The qualities of sdfThing are shown in Table 10. None of these qualities are required or have default values that are assumed if the quality is absent. Analogous to sdfObject, the presence of one or both of the optional qualities "`minItems`" and "`maxItems`" defines the sdfThing as an array.

Quality	Type	Description
(common)		Section 4.6
sdfThing	thing	
sdfObject	object	
sdfProperty	property	zero or more named property definitions for this thing
sdfAction	action	zero or more named action definitions for this thing
sdfEvent	event	zero or more named event definitions for this thing

Quality	Type	Description
sdfData	named-sdq	zero or more named data type definitions that might be used in the above
minItems	number	(array) minimum number of multiplied affordances in array
maxItems	number	(array) maximum number of multiplied affordances in array

Table 10: Qualities of sdfThing

7. IANA Considerations

7.1. Media Type

IANA has added the following Media-Type to the "Media Types" registry [IANA.media-types].

Name	Template	Reference
sdf+json	application/sdf+json	RFC 9880, Section 7.1

Table 11: Media Type Registration for SDF

Type name: application

Subtype name: sdf+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary (JSON is UTF-8-encoded text)

Security considerations: Section 8 of RFC 9880

Interoperability considerations: none

Published specification: Section 7.1 of RFC 9880

Applications that use this media type: Tools for data and interaction modeling in the Internet of Things and related environments.

Fragment identifier considerations: A JSON Pointer fragment identifier may be used as defined in Section 6 of [RFC6901].

Additional information:

Magic number(s): n/a
File extension(s): .sdf.json
Windows Clipboard Name: "Semantic Definition Format (SDF) for Data and Interactions of Things"

Macintosh file type code(s): n/a
Macintosh Universal Type Identifier code: org.ietf.sdf-json conforms to public.text

Person & email address to contact for further information: ASDF WG mailing list (asdf@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IETF

Provisional registration: no

7.2. Content-Format

IANA has added the following Content-Format to the "CoAP Content-Formats" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group [IANA.core-parameters].

Content Type	Content Coding	ID	Reference
application/sdf+json	-	434	RFC 9880

Table 12: SDF Content-Format Registration

7.3. IETF URN Sub-Namespace for Unit Names (urn:ietf:params:unit)

IANA has registered the following value in the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry in [IANA.params], following the template in [BCP73]:

Registry name: unit

Specification: RFC 9880

Repository: Combining the symbol values from the "SenML Units" registry and the "Secondary Units" registry in the "Sensor Measurement Lists (SenML)" registry group [IANA.senml] as specified by Sections 4.5.2 and 12.1 of [RFC8428] and Section 3 of [RFC8798], respectively (which, by the registration policy, are guaranteed to be non-overlapping).

Index value: Percent-encoding (Section 2.1 of RFC 3986 [STD66]) is required of any characters in unit names except for the set "`unreserved`" (Section 2.3 of RFC 3986 [STD66]), the set "`sub-delims`" (Section 2.2 of RFC 3986 [STD66]), and "`:`" or "`@`" (i.e., the result must match the ABNF rule "`pchar`" in Section 3.3 of RFC 3986 [STD66]).

7.4. SenML Registry Group

IANA has added the following note to the "Sensor Measurement Lists (SenML)" registry group [IANA.senml]:

> In SDF [RFC9880], a URI unit name is distinguished from a registered unit name by the presence of a colon; any registered unit name that contains a colon can therefore not be directly used in SDF.

7.5. Registries

IANA has created the "Semantic Definition Format (SDF)" registry group with the registries defined in this Section.

7.5.1. SDF Quality Names

IANA has created the "SDF Quality Names" registry in the "Semantic Definition Format (SDF)" registry group with the following template:

Name: A Quality Name composed of ASCII letters, digits, and dollar signs, starting with a lowercase ASCII letter or a dollar sign (i.e., using a pattern of "`[a-z$][A-Za-z$0-9]*`").

Brief Description: A brief description.

Reference: A pointer to a specification.

Change Controller: (See Section 2.3 of RFC 8126 [BCP26])

Quality Names in this registry are intended to be registered in conjunction with RFCs and activities of the IETF.

The registration policy is Specification Required as per Section 4.6 of RFC 8126 [BCP26]. Note that the policy is not "RFC Required" or "IETF Review" (Sections 4.7 and 4.8 of RFC 8126 [BCP26]) so that registrations can be made earlier in the process, even earlier than foreseen in [BCP100].)

The instructions to the Experts are:

- to ascertain that the specification is available in an immutable reference and has achieved a good level of review in conjunction with RFCs or activities of the IETF, and
- to be frugal in the allocation of Quality Names that are suggestive of generally applicable semantics, keeping them in reserve for qualities that are likely to enjoy wide use and can make good use of their conciseness.

The "SDF Quality Names" registry starts out as in Table 13; all references for these initial entries are to RFC 9880 (this document) and all change controllers are "IETF".

Name	Brief Description
$comment	source code comments only, no semantics
const	constant value

Name	Brief Description
contentFormat	content format
default	default value
description	long description text
enum	sdfChoice limited to text strings
exclusiveMaximum	exclusive maximum for a number
exclusiveMinimum	exclusive minimum for a number
format	specific format for a text string
items	items of an array
label	short text (no constraints); defaults to key
maxItems	maximum number of items in an array
maxLength	maximum length for a text string (in characters, i.e., Unicode scalar values)
maximum	maximum for a number
minItems	minimum number of items in an array
minLength	minimum length for a text string (in characters, i.e., Unicode scalar values)
minimum	minimum for a number
multipleOf	step size of number
nullable	boolean: can the item be left out?
observable	boolean: can the item be observed?
pattern	regular expression pattern for a text string
properties	named dataqualities for type="object"
readable	boolean: can the item be read?
required	which data items are required?
sdfChoice	named dataqualities for a choice

Name	Brief Description
sdfData	named dataqualities for an independent data type definition
sdfInputData	input data to an action
sdfOutputData	output data of an action or event (sdfRequired applies here)
sdfRef	sdf-pointer to definition being referenced
sdfRequired	pointer-list to declarations of required components
sdfRequiredInputData	pointer-list to declarations of required input data for an action
sdfType	more detailed information about the type of a string
type	general category of data type
uniqueItems	boolean: do the items of the array need to be all different?
unit	engineering unit and scale (per SenML registry)
writable	boolean: can the item be written to?

Table 13: Initial Content of the SDF Quality Names Registry

7.5.2. SDF Quality Name Prefixes

IANA has created the "SDF Quality Name Prefixes" registry in the "Semantic Definition Format (SDF)" registry group with the following template:

Prefix: A Quality Name prefix composed of lowercase ASCII letters and digits, starting with a lowercase ASCII letter (i.e., using a pattern of "`[a-z][a-z0-9]*`").

Contact: A contact point for the organization that assigns Quality Names with this prefix.

Reference: A pointer to additional information, if available.

Quality Name Prefixes are intended to be registered by organizations that plan to define Quality Names constructed with an organization-specific prefix (Section 2.3.3).

The registration policy is Expert Review as per Section 4.5 of RFC 8126 [BCP26]. The instructions to the Expert are to ascertain that the organization will handle Quality Names constructed using their prefix in a way that roughly achieves the objectives for an IANA registry that supports interoperability of SDF models employing these Quality Names, including:

- Stability, "stable and permanent";
- Transparency, "readily available" and "in sufficient detail" (Section 4.6 of RFC 8126 [BCP26]).

The "SDF Quality Name Prefixes" registry is empty at this time.

7.5.3. sdfType Values

IANA has created the "sdfType Values" registry in the "Semantic Definition Format (SDF)" registry group with the following template:

Name: A name composed of lowercase ASCII letters, digits and - (ASCII hyphen/minus) characters, starting with a lowercase ASCII letter (i.e., using a pattern of "`[a-z][-a-z0-9]*`").

Description: A short description of the information model level structure and semantics.

type: The value of the quality "type" to be used with this sdfType.

JSON Representation: A short description of a JSON representation that can be used for this sdfType. As per Section 4.7.1, this **MUST** be consistent with the type.

Reference: A more detailed specification of meaning and use of sdfType.

sdfType values are intended to be registered to enable modeling additional SDF-specific types (see Section 4.7.1).

The registration policy is Specification Required as per Section 4.6 of RFC 8126 [BCP26]. The instructions to the Expert are to ascertain that the specification provides enough detail to enable interoperability between implementations of the sdfType being registered, and that names are chosen with enough specificity that ecosystem-specific sdfTypes will not be confused with more generally applicable ones.

The initial set of registrations is described in Table 5.

7.5.4. SDF Feature Names

IANA has created the "SDF Feature Names" registry in the "Semantic Definition Format (SDF)" registry group with the following template:

Name: A feature name composed of ASCII letters, digits, and dollar signs, starting with a lowercase ASCII letter or a dollar sign (i.e., using a pattern of "`[a-z$][A-Za-z$0-9]*`").

Brief Description: A brief description.

Reference: A pointer to a specification.

Change Controller: (See Section 2.3 of RFC 8126 [BCP26])

The registration policy is Specification Required as per Section 4.6 of RFC 8126 [BCP26].

The instructions to the Experts are:

- to ascertain that the specification is available in an immutable reference and has achieved a good level of review, and

- to be frugal in the allocation of feature names that are suggestive of generally applicable semantics, keeping them in reserve for features that are likely to enjoy wide use and can make good use of their conciseness.

The "SDF Feature Names" registry is empty at this time.

8. Security Considerations

Some wider security considerations applicable to Things are discussed in [RFC8576].

Section 5 of [RFC8610] gives an overview over security considerations that arise when formal description techniques are used to govern interoperability; analogs of these security considerations can apply to SDF.

The security considerations of underlying building blocks such as those detailed in Section 10 of RFC 3629 [STD63] apply.

SDF uses JSON as a representation language. For a number of cases, [STD90] indicates that implementation behavior for certain constructs allowed by the JSON grammar is unpredictable.

Implementations need to be robust against invalid or unpredictable cases on input, preferably by rejecting input that is invalid or that would lead to unpredictable behavior, and avoid generating these cases on output.

Implementations of model languages may also exhibit performance-related availability issues when the attacker can control the input, see Section 4.1 of [RFC9535] for a brief discussion and Section 8 of [RFC9485] for considerations specific to the use of `pattern`.

SDF may be used in two processes that are often security relevant: (1) model-based *validation* of data that is intended to be described by SDF models, and (2) model-based *augmentation* of these data with information obtained from the SDF models that apply.

Implementations need to ascertain the provenance (and thus authenticity and integrity) and applicability of the SDF models they employ operationally in such security-relevant ways. Implementations that make use of the composition mechanisms defined in this document need to do this for each of the components they combine into the SDF models they employ. Essentially, this process needs to undergo the same care and scrutiny as any other introduction of source code into a build environment; the possibility of supply-chain attacks on the modules imported needs to be considered.

Specifically, implementations might rely on model-based input validation for enforcing certain characteristics of the data structure ingested (which, if not validated, could lead to malfunctions such as crashes and remote code execution). These implementations need to be particularly careful about the data models they apply, including their provenance and potential changes of these characteristics that upgrades to the referenced modules may (inadvertently or as part of an attack) cause. More generally speaking, implementations should strive to be robust against expected and unexpected limitations of the model-based input validation mechanisms and their implementations.

Similarly, implementations that rely on model-based augmentation may generate false data from their inputs; this is an integrity violation in any case, but also can possibly be exploited for further attacks.

In applications that dynamically acquire models and obtain modules from module references in these models, the security considerations of dereferenceable identifiers apply (see [DEREF-ID-PATTERN] for a more extensive discussion of dereferenceable identifiers).

There may be confidentiality requirements on SDF models, both on their content and on the fact that a specific model is used in a particular Thing or environment. The present specification does not discuss model discovery or define an access control model for SDF models, nor does it define a way to obtain selective disclosure where that may be required. It is likely that these definitions require additional context from underlying ecosystems and the characteristics of the protocols employed there; therefore, this is left as future work (e.g., for documents such as [SDF-MAPPING]).

9. References

9.1. Normative References

[BCP26] Best Current Practice 26, <https://www.rfc-editor.org/info/bcp26>.
At the time of writing, this BCP comprises the following:

Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[BCP73] Best Current Practice 73, <https://www.rfc-editor.org/info/bcp73>.
At the time of writing, this BCP comprises the following:

Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://www.rfc-editor.org/info/rfc3553>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[IANA.params] IANA, "Uniform Resource Name (URN) Namespace for IETF Use", <https://www.iana.org/assignments/params>.

[IANA.senml] IANA, "Sensor Measurement Lists (SenML)", <https://www.iana.org/assignments/senml>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/info/rfc6901>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <https://www.rfc-editor.org/info/rfc7396>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <https://www.rfc-editor.org/info/rfc8428>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8798] Bormann, C., "Additional Units for Sensor Measurement Lists (SenML)", RFC 8798, DOI 10.17487/RFC8798, June 2020, <https://www.rfc-editor.org/info/rfc8798>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/info/rfc9165>.

[RFC9193] Keränen, A. and C. Bormann, "Sensor Measurement Lists (SenML) Fields for Indicating Data Value Content-Format", RFC 9193, DOI 10.17487/RFC9193, June 2022, <https://www.rfc-editor.org/info/rfc9193>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[SPDX] "SPDX License List", <https://spdx.org/licenses/>.

[STD63] Internet Standard 63, <https://www.rfc-editor.org/info/std63>.
At the time of writing, this STD comprises the following:

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[STD66] Internet Standard 66, <https://www.rfc-editor.org/info/std66>.
At the time of writing, this STD comprises the following:

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[STD80] Internet Standard 80, <https://www.rfc-editor.org/info/std80>.
At the time of writing, this STD comprises the following:

Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[STD90] Internet Standard 90, <https://www.rfc-editor.org/info/std90>.
At the time of writing, this STD comprises the following:

Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[STD94] Internet Standard 94, <https://www.rfc-editor.org/info/std94>.
At the time of writing, this STD comprises the following:

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[W3C.NOTE-curie-20101216] Birbeck, M., Ed. and S. McCarron, Ed., "CURIE Syntax 1.0", W3C Working Group Note, 16 December 2010, <https://www.w3.org/TR/2010/NOTE-curie-20101216/>.

9.2. Informative References

[BCP100] Best Current Practice 100, <https://www.rfc-editor.org/info/bcp100>.
At the time of writing, this BCP comprises the following:

Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[CamelCase] "Camel Case", December 2014, <http://wiki.c2.com/?CamelCase>.

[DEREF-ID-PATTERN] Bormann, C. and C. Amsüss, "The "dereferenceable identifier" pattern", Work in Progress, Internet-Draft, draft-bormann-t2trg-deref-id-06, 30 August 2025, <https://datatracker.ietf.org/doc/html/draft-bormann-t2trg-deref-id-06>.

[ECMA-262] Ecma International, "ECMAScript 2025 Language Specification", 16th Edition, ECMA Standard ECMA-262, June 2025, <https://ecma-international.org/wp-content/uploads/ECMA-262_16th_edition_june_2025.pdf>.

[JSO4] Galiegue, F., Ed., Zyp, K., Ed., and G. Court, "JSON Schema: core definitions and terminology", Work in Progress, Internet-Draft, draft-zyp-json-schema-04, 31 January 2013, <https://datatracker.ietf.org/doc/html/draft-zyp-json-schema-04>.

[JSO4V] Zyp, K. and G. Court, "JSON Schema: interactive and non interactive validation", Work in Progress, Internet-Draft, draft-fge-json-schema-validation-00, 31 January 2013, <https://datatracker.ietf.org/doc/html/draft-fge-json-schema-validation-00>.

[JSO7] Wright, A., Ed., Andrews, H., Ed., Hutton, B., Ed., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", Work in Progress, Internet-Draft, draft-handrews-json-schema-02, 17 September 2019, <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-02>.

[JSO7V] Wright, A., Ed., Andrews, H., Ed., and B. Hutton, Ed., "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", Work in Progress, Internet-Draft, draft-handrews-json-schema-validation-02, 17 September 2019, <https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-validation-02>.

[KebabCase] "Kebab Case", August 2014, <http://wiki.c2.com/?KebabCase>.

[OCF] Open Connectivity Foundation, "OCF Resource Type Specification", Version 2.2.7, November 2023, <https://openconnectivity.org/specs/OCF_Resource_Type_Specification.pdf>.

[OMA] Open Mobile Alliance, "LwM2M OBJECTS", <https://www.openmobilealliance.org/specifications/registries/objects>.

[REST-IOT] Keränen, A., Kovatsch, M., and K. Hartke, "Guidance on RESTful Design for Internet of Things Systems", Work in Progress, Internet-Draft, draft-irtf-t2trg-rest-iot-17, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-rest-iot-17>.

[RFC8576] Garcia-Morchon, O., Kumar, S., and M. Sethi, "Internet of Things (IoT) Security: State of the Art and Challenges", RFC 8576, DOI 10.17487/RFC8576, April 2019, <https://www.rfc-editor.org/info/rfc8576>.

[RFC9485] Bormann, C. and T. Bray, "I-Regexp: An Interoperable Regular Expression Format", RFC 9485, DOI 10.17487/RFC9485, October 2023, <https://www.rfc-editor.org/info/rfc9485>.

[RFC9535] Gössner, S., Ed., Normington, G., Ed., and C. Bormann, Ed., "JSONPath: Query Expressions for JSON", RFC 9535, DOI 10.17487/RFC9535, February 2024, <https://www.rfc-editor.org/info/rfc9535>.

[SDF-MAPPING] Bormann, C. and J. Romann, "Semantic Definition Format (SDF): Mapping files", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-mapping-00, 18 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-mapping-00>.

[SDFTYPE-LINK] Bormann, C. and A. Keränen, "An sdfType for Links", Work in Progress, Internet-Draft, draft-ietf-asdf-sdftype-link-01, 19 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdftype-link-01>.

[STD97] Internet Standard 97, <https://www.rfc-editor.org/info/std97>.
At the time of writing, this STD comprises the following:

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[WoT] Kaebisch, S., Ed., McCool, M., Ed., and E. Korkan, Ed., "Web of Things (WoT) Thing Description 1.1", W3C Recommendation, 5 December 2023, <https://www.w3.org/TR/2023/REC-wot-thing-description11-20231205/>.

[ZCL] "Chapter 6 - The ZigBee Cluster Library", Zigbee Wireless Networking, pp. 239-271, DOI 10.1016/b978-0-7506-8597-9.00006-9, ISBN 9780750685979, 2008, <https://doi.org/10.1016/b978-0-7506-8597-9.00006-9>.

Appendix A. Formal Syntax of SDF

This normative appendix describes the syntax of SDF using CDDL [RFC8610].

This appendix shows the framework syntax only, i.e., a syntax with liberal extension points. Since this syntax is nearly useless in finding typos in an SDF specification, a second syntax, the validation syntax, is defined that does not include the extension points. The validation syntax can be generated from the framework syntax by leaving out all lines containing the string `EXTENSION-POINT`; as this is trivial, the result is not shown here.

This appendix makes use of CDDL "features" as defined in Section 4 of [RFC9165]. Features whose names end in "-ext" indicate extension points for further evolution.

```
start = sdf-syntax

sdf-syntax = {
 ; info will be required in most process policies
 ? info: sdfinfo
 ? namespace: named<text>
 ? defaultNamespace: text
 ; Thing is a composition of objects that work together in some way
 ? sdfThing: named<thingqualities>
 ; Object is a set of Properties, Actions, and Events that together
 ; perform a particular function
 ? sdfObject: named<objectqualities>
```

```
 ; Includes Properties, Actions, and Events as well as sdfData
 paedataqualities
 * $$SDF-EXTENSION-TOP
 EXTENSION-POINT<"top-ext">
}

sdfinfo = {
 ? title: text
 ? description: text
 ? version: text
 ? copyright: text
 ? license: text
 ? modified: modified-date-time
 ? features: [
               * (any .feature "feature-name") ; EXTENSION-POINT
             ]
 optional-comment
 * $$SDF-EXTENSION-INFO
 EXTENSION-POINT<"info-ext">
}

; Shortcut for a map that gives names to instances of X
; (has keys of type text and values of type X)
named<X> = { * text => X }

; EXTENSION-POINT is only used in framework syntax
EXTENSION-POINT<f> = ( * (quality-name .feature f) => any )
quality-name = text .regexp "([a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*"

sdf-pointer = global / same-object / true
global = text .regexp ".*[:#].*" ; rough CURIE or JSON Pointer syntax
same-object = referenceable-name
referenceable-name = text .regexp "[^:#]*"

; per se no point in having an empty list, but used for sdfRequired
; in odmobject-multiple_axis_joystick.sdf.json
pointer-list = [* sdf-pointer]

optional-comment = (
 ? $comment: text       ; source code comments only, no semantics
)

commonqualities = (
 ? description: text    ; long text (no constraints)
 ? label: text          ; short text (no constraints); default to key
 optional-comment
 ? sdfRef: sdf-pointer
 ; applies to qualities of properties, of data:
 ? sdfRequired: pointer-list
)

arraydefinitionqualities = (
 ? "minItems" => uint
 ? "maxItems" => uint
)

paedataqualities = (
 ; Property represents the state of an instance of an object
```

```
 ? sdfProperty: named<propertyqualities>
 ; Action invokes an application layer verb associated with an object
 ? sdfAction: named<actionqualities>
 ; Event represents an occurrence of event associated with an object
 ? sdfEvent: named<eventqualities>
 ; Data represents a piece of information that can be the state of a
 ; property or a parameter to an action or a signal in an event
 ? sdfData: named<dataqualities>

)

; for building hierarchy
thingqualities = {
 commonqualities
 ? sdfObject: named<objectqualities>
 ? sdfThing: named<thingqualities>
 paedataqualities
 arraydefinitionqualities
 * $$SDF-EXTENSION-THING
 EXTENSION-POINT<"thing-ext">
}

; for single objects, or for arrays of objects
objectqualities = {
 commonqualities
 paedataqualities
 arraydefinitionqualities
 * $$SDF-EXTENSION-OBJECT
 EXTENSION-POINT<"object-ext">
}

parameter-list = dataqualities

actionqualities = {
 commonqualities
 ? sdfInputData: parameter-list   ; sdfRequiredInputData applies here
 ? sdfOutputData: parameter-list  ; sdfRequired applies here
 ; zero or more named data type definitions that might be used above
 ? sdfData: named<dataqualities>
 * $$SDF-EXTENSION-ACTION
 EXTENSION-POINT<"action-ext">
}

eventqualities = {
 commonqualities
 ? sdfOutputData: parameter-list  ; sdfRequired applies here
 ; zero or more named data type definitions that might be used above
 ? sdfData: named<dataqualities>
 * $$SDF-EXTENSION-EVENT
 EXTENSION-POINT<"event-ext">
}

sdftype-name = text .regexp "[a-z][-a-z0-9]*"       ; EXTENSION-POINT

dataqualities = {
 commonqualities
 jsonschema
 ? "unit" => text
```

```
 ? nullable: bool
 ? "sdfType" => "byte-string" / "unix-time"
            / $SDF-EXTENSION-SDFTYPE .within sdftype-name
            / (sdftype-name .feature "sdftype-ext") ; EXTENSION-POINT
 ? contentFormat: text
 * $$SDF-EXTENSION-DATA
 EXTENSION-POINT<"data-ext">
}

propertyqualities = {
 ? observable: bool
 ? readable: bool
 ? writable: bool
 * $$SDF-EXTENSION-PROPERTY
 ~dataqualities
}

allowed-types = number / text / bool / null
              / [* number] / [* text] / [* bool]
              / {* text => any}
              / $SDF-EXTENSION-ALLOWED
              / (any .feature "allowed-ext")        ; EXTENSION-POINT

compound-type = (
  "type" => "object"
  ? required: [+text]
  ? properties: named<dataqualities>
)

optional-choice = (
 ? (("sdfChoice" => named<dataqualities>)
  // ("enum" => [+ text])) ; limited to text strings
)

jsonschema = (
 ? (("type" => "number" / "string" / "boolean" / "integer" / "array")
    // compound-type
    // $$SDF-EXTENSION-TYPE
    // (type: text .feature "type-ext")             ; EXTENSION-POINT
   )
 ; if present, all other qualities apply to all choices:
 optional-choice
 ; the next three should validate against type:
 ? const: allowed-types
 ? default: allowed-types
 ; number/integer constraints
 ? minimum: number
 ? maximum: number
 ? exclusiveMinimum: number
 ? exclusiveMaximum: number
 ? multipleOf: number
 ; text string constraints
 ? minLength: uint
 ? maxLength: uint
 ? pattern: text                ; regexp
 ? format: "date-time" / "date" / "time"
           / "uri" / "uri-reference" / "uuid"
           / $SDF-EXTENSION-FORMAT .within text
```

```
           / (text .feature "format-ext")          ; EXTENSION-POINT
 ; array constraints
 ? minItems: uint
 ? maxItems: uint
 ? uniqueItems: bool
 ? items: jso-items
)

jso-items = {
     ? sdfRef: sdf-pointer ; import limited to subset allowed here...
     ? description: text   ; long text (no constraints)
     optional-comment
     ; leave commonqualities out for non-complex data types,
     ; but need the above three.
     ; no further nesting: no "array"
     ? ((type: "number" / "string" / "boolean" / "integer")
        // compound-type
        // $$SDF-EXTENSION-ITEMTYPE
        // (type: text .feature "itemtype-ext")     ; EXTENSION-POINT
       )
     ; if present, all other qualities apply to all choices
     optional-choice
     ; jso subset
     ? minimum: number
     ? maximum: number
     ? format: text
     ? minLength: uint
     ? maxLength: uint
     * $$SDF-EXTENSION-ITEMS
     EXTENSION-POINT<"items-ext">
   }

modified-date-time = text .abnf modified-dt-abnf
modified-dt-abnf = "modified-dt" .det rfc3339z

; RFC 3339 sans time-numoffset, slightly condensed
rfc3339z = '
   date-fullyear   = 4DIGIT
   date-month      = 2DIGIT  ; 01-12
   date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on
                             ; month/year
   time-hour       = 2DIGIT  ; 00-23
   time-minute     = 2DIGIT  ; 00-59
   time-second     = 2DIGIT  ; 00-58, 00-59, 00-60 based on leap sec
                             ; rules
   time-secfrac    = "." 1*DIGIT
   DIGIT           =  %x30-39 ; 0-9

   partial-time    = time-hour ":" time-minute ":" time-second
                     [time-secfrac]
   full-date       = date-fullyear "-" date-month "-" date-mday

   modified-dt     = full-date ["T" partial-time "Z"]
'
```

Appendix B. json-schema.org Rendition of SDF Syntax

This informative appendix describes the syntax of SDF defined in Appendix A, but uses a version of the description techniques advertised on json-schema.org [JSO7] [JSO7V].

The appendix shows both the validation and the framework syntax. Since most of the lines are the same between these two files, those lines are shown only once, with a leading space, in the form of a unified diff. Lines leading with a `-` are part of the validation syntax and lines leading with a `+` are part of the framework syntax.

```
 {
-  "title": "sdf-validation.cddl -- Generated: 2025-10-13T08:43:18Z",
+  "title": "sdf-framework.cddl -- Generated: 2025-10-13T08:43:29Z",
   "$schema": "http://json-schema.org/draft-07/schema#",
   "$ref": "#/definitions/sdf-syntax",
   "definitions": {
     "sdf-syntax": {
       "type": "object",
       "properties": {
         "info": {
           "$ref": "#/definitions/sdfinfo"
         },
         "namespace": {
           "type": "object",
           "additionalProperties": {
             "type": "string"
           }
         },
         "defaultNamespace": {
           "type": "string"
         },
         "sdfThing": {
           "type": "object",
           "additionalProperties": {
             "$ref": "#/definitions/thingqualities"
           }
         },
         "sdfObject": {
           "type": "object",
           "additionalProperties": {
             "$ref": "#/definitions/objectqualities"
           }
         },
         "sdfProperty": {
           "$ref": "#/definitions/sdfProperty-"
         },
         "sdfAction": {
           "$ref": "#/definitions/sdfAction-"
         },
         "sdfEvent": {
           "$ref": "#/definitions/sdfEvent-"
         },
         "sdfData": {
```

```
            "$ref": "#/definitions/sdfData-sdfChoice-properties-"
          }
        },
+       "patternProperties": {
+         "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+       },
        "additionalProperties": false
      },
      "sdfinfo": {
        "type": "object",
        "properties": {
          "title": {
            "type": "string"
          },
          "description": {
            "type": "string"
          },
          "version": {
            "type": "string"
          },
          "copyright": {
            "type": "string"
          },
          "license": {
            "type": "string"
          },
          "modified": {
            "$ref": "#/definitions/modified-date-time"
          },
          "features": {
-           "type": "array",
-           "maxItems": 0
+           "type": "array"
          },
          "$comment": {
            "type": "string"
          }
        },
+       "patternProperties": {
+         "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+       },
        "additionalProperties": false
      },
      "modified-date-time": {
        "type": "string"
      },
      "thingqualities": {
        "type": "object",
        "properties": {
          "description": {
            "type": "string"
          },
          "label": {
            "type": "string"
          },
          "$comment": {
            "type": "string"
          },
```

```
          "sdfRef": {
            "$ref": "#/definitions/sdf-pointer"
          },
          "sdfRequired": {
            "$ref": "#/definitions/pointer-list"
          },
          "sdfObject": {
            "type": "object",
            "additionalProperties": {
              "$ref": "#/definitions/objectqualities"
            }
          },
          "sdfThing": {
            "type": "object",
            "additionalProperties": {
              "$ref": "#/definitions/thingqualities"
            }
          },
          "sdfProperty": {
            "$ref": "#/definitions/sdfProperty-"
          },
          "sdfAction": {
            "$ref": "#/definitions/sdfAction-"
          },
          "sdfEvent": {
            "$ref": "#/definitions/sdfEvent-"
          },
          "sdfData": {
            "$ref": "#/definitions/sdfData-sdfChoice-properties-"
          },
          "minItems": {
            "$ref": "#/definitions/uint"
          },
          "maxItems": {
            "$ref": "#/definitions/uint"
          }
        },
+       "patternProperties": {
+         "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+       },
        "additionalProperties": false
      },
      "sdf-pointer": {
        "anyOf": [
          {
            "$ref": "#/definitions/global"
          },
          {
            "$ref": "#/definitions/same-object"
          },
          {
            "$ref": "#/definitions/true"
          }
        ]
      },
      "global": {
        "type": "string",
        "pattern": "^[^\\n\\r]*[:#][^\\n\\r]*$"
```

```
      },
      "same-object": {
        "$ref": "#/definitions/referenceable-name"
      },
      "referenceable-name": {
        "type": "string",
        "pattern": "^[^:#]*$"
      },
      "true": {
        "type": "boolean",
        "const": true
      },
      "pointer-list": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/sdf-pointer"
        }
      },
      "objectqualities": {
        "type": "object",
        "properties": {
          "description": {
            "type": "string"
          },
          "label": {
            "type": "string"
          },
          "$comment": {
            "type": "string"
          },
          "sdfRef": {
            "$ref": "#/definitions/sdf-pointer"
          },
          "sdfRequired": {
            "$ref": "#/definitions/pointer-list"
          },
          "sdfProperty": {
            "$ref": "#/definitions/sdfProperty-"
          },
          "sdfAction": {
            "$ref": "#/definitions/sdfAction-"
          },
          "sdfEvent": {
            "$ref": "#/definitions/sdfEvent-"
          },
          "sdfData": {
            "$ref": "#/definitions/sdfData-sdfChoice-properties-"
          },
          "minItems": {
            "$ref": "#/definitions/uint"
          },
          "maxItems": {
            "$ref": "#/definitions/uint"
          }
        },
+       "patternProperties": {
+         "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+       },
```

```
        "additionalProperties": false
      },
      "propertyqualities": {
        "anyOf": [
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "$ref": "#/definitions/type-"
              },
              "sdfChoice": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "observable": {
                "type": "boolean"
              },
              "readable": {
                "type": "boolean"
              },
              "writable": {
                "type": "boolean"
              },
              "description": {
                "type": "string"
              },
              "label": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "sdfRequired": {
                "$ref": "#/definitions/pointer-list"
              },
              "const": {
                "$ref": "#/definitions/allowed-types"
              },
              "default": {
                "$ref": "#/definitions/allowed-types"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "exclusiveMinimum": {
                "type": "number"
              },
              "exclusiveMaximum": {
                "type": "number"
              },
```

```
              "multipleOf": {
                "type": "number"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              },
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "const": "object"
              },
              "required": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
```

```
            },
            "properties": {
              "$ref": "#/definitions/sdfData-sdfChoice-properties-"
            },
            "sdfChoice": {
              "$ref": "#/definitions/sdfData-sdfChoice-properties-"
            },
            "observable": {
              "type": "boolean"
            },
            "readable": {
              "type": "boolean"
            },
            "writable": {
              "type": "boolean"
            },
            "description": {
              "type": "string"
            },
            "label": {
              "type": "string"
            },
            "$comment": {
              "type": "string"
            },
            "sdfRef": {
              "$ref": "#/definitions/sdf-pointer"
            },
            "sdfRequired": {
              "$ref": "#/definitions/pointer-list"
            },
            "const": {
              "$ref": "#/definitions/allowed-types"
            },
            "default": {
              "$ref": "#/definitions/allowed-types"
            },
            "minimum": {
              "type": "number"
            },
            "maximum": {
              "type": "number"
            },
            "exclusiveMinimum": {
              "type": "number"
            },
            "exclusiveMaximum": {
              "type": "number"
            },
            "multipleOf": {
              "type": "number"
            },
            "minLength": {
              "$ref": "#/definitions/uint"
            },
            "maxLength": {
              "$ref": "#/definitions/uint"
            },
```

```
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
+           "properties": {
+             "type": {
+               "type": "string"
+             },
+             "sdfChoice": {
+               "$ref": "#/definitions/sdfData-sdfChoice-properties-"
+             },
+             "observable": {
+               "type": "boolean"
+             },
+             "readable": {
+               "type": "boolean"
+             },
+             "writable": {
+               "type": "boolean"
+             },
+             "description": {
+               "type": "string"
+             },
+             "label": {
```

```
+              "type": "string"
+            },
+            "$comment": {
+              "type": "string"
+            },
+            "sdfRef": {
+              "$ref": "#/definitions/sdf-pointer"
+            },
+            "sdfRequired": {
+              "$ref": "#/definitions/pointer-list"
+            },
+            "const": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "default": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "minimum": {
+              "type": "number"
+            },
+            "maximum": {
+              "type": "number"
+            },
+            "exclusiveMinimum": {
+              "type": "number"
+            },
+            "exclusiveMaximum": {
+              "type": "number"
+            },
+            "multipleOf": {
+              "type": "number"
+            },
+            "minLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "pattern": {
+              "type": "string"
+            },
+            "format": {
+              "$ref": "#/definitions/format-"
+            },
+            "minItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "uniqueItems": {
+              "type": "boolean"
+            },
+            "items": {
+              "$ref": "#/definitions/jso-items"
+            },
+            "unit": {
+              "type": "string"
```

```
+             },
+             "nullable": {
+               "type": "boolean"
+             },
+             "sdfType": {
+               "$ref": "#/definitions/sdfType-"
+             },
+             "contentFormat": {
+               "type": "string"
+             }
+           },
+           "additionalProperties": false
+         },
+         {
+           "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
             "properties": {
               "type": {
                 "$ref": "#/definitions/type-"
               },
               "enum": {
                 "type": "array",
                 "items": {
                   "type": "string"
                 },
                 "minItems": 1
               },
               "observable": {
                 "type": "boolean"
               },
               "readable": {
                 "type": "boolean"
               },
               "writable": {
                 "type": "boolean"
               },
               "description": {
                 "type": "string"
               },
               "label": {
                 "type": "string"
               },
               "$comment": {
                 "type": "string"
               },
               "sdfRef": {
                 "$ref": "#/definitions/sdf-pointer"
               },
               "sdfRequired": {
                 "$ref": "#/definitions/pointer-list"
               },
               "const": {
                 "$ref": "#/definitions/allowed-types"
               },
               "default": {
                 "$ref": "#/definitions/allowed-types"
```

```
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "exclusiveMinimum": {
                "type": "number"
              },
              "exclusiveMaximum": {
                "type": "number"
              },
              "multipleOf": {
                "type": "number"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              },
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
```

```
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "const": "object"
              },
              "required": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "properties": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "enum": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "observable": {
                "type": "boolean"
              },
              "readable": {
                "type": "boolean"
              },
              "writable": {
                "type": "boolean"
              },
              "description": {
                "type": "string"
              },
              "label": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "sdfRequired": {
                "$ref": "#/definitions/pointer-list"
              },
              "const": {
                "$ref": "#/definitions/allowed-types"
              },
              "default": {
                "$ref": "#/definitions/allowed-types"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
```

```
                "type": "number"
              },
              "exclusiveMinimum": {
                "type": "number"
              },
              "exclusiveMaximum": {
                "type": "number"
              },
              "multipleOf": {
                "type": "number"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              },
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
+         },
+         {
+           "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
+           "properties": {
+             "type": {
+               "type": "string"
```

```
+            },
+            "enum": {
+              "type": "array",
+              "items": {
+                "type": "string"
+              },
+              "minItems": 1
+            },
+            "observable": {
+              "type": "boolean"
+            },
+            "readable": {
+              "type": "boolean"
+            },
+            "writable": {
+              "type": "boolean"
+            },
+            "description": {
+              "type": "string"
+            },
+            "label": {
+              "type": "string"
+            },
+            "$comment": {
+              "type": "string"
+            },
+            "sdfRef": {
+              "$ref": "#/definitions/sdf-pointer"
+            },
+            "sdfRequired": {
+              "$ref": "#/definitions/pointer-list"
+            },
+            "const": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "default": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "minimum": {
+              "type": "number"
+            },
+            "maximum": {
+              "type": "number"
+            },
+            "exclusiveMinimum": {
+              "type": "number"
+            },
+            "exclusiveMaximum": {
+              "type": "number"
+            },
+            "multipleOf": {
+              "type": "number"
+            },
+            "minLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxLength": {
+              "$ref": "#/definitions/uint"
```

```
+            },
+            "pattern": {
+              "type": "string"
+            },
+            "format": {
+              "$ref": "#/definitions/format-"
+            },
+            "minItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "uniqueItems": {
+              "type": "boolean"
+            },
+            "items": {
+              "$ref": "#/definitions/jso-items"
+            },
+            "unit": {
+              "type": "string"
+            },
+            "nullable": {
+              "type": "boolean"
+            },
+            "sdfType": {
+              "$ref": "#/definitions/sdfType-"
+            },
+            "contentFormat": {
+              "type": "string"
+            }
+          },
+          "additionalProperties": false
        }
      ]
    },
    "dataqualities": {
      "anyOf": [
        {
          "type": "object",
+          "patternProperties": {
+            "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+          },
          "properties": {
            "type": {
              "$ref": "#/definitions/type-"
            },
            "sdfChoice": {
              "$ref": "#/definitions/sdfData-sdfChoice-properties-"
            },
            "description": {
              "type": "string"
            },
            "label": {
              "type": "string"
            },
            "$comment": {
              "type": "string"
```

```
            },
            "sdfRef": {
              "$ref": "#/definitions/sdf-pointer"
            },
            "sdfRequired": {
              "$ref": "#/definitions/pointer-list"
            },
            "const": {
              "$ref": "#/definitions/allowed-types"
            },
            "default": {
              "$ref": "#/definitions/allowed-types"
            },
            "minimum": {
              "type": "number"
            },
            "maximum": {
              "type": "number"
            },
            "exclusiveMinimum": {
              "type": "number"
            },
            "exclusiveMaximum": {
              "type": "number"
            },
            "multipleOf": {
              "type": "number"
            },
            "minLength": {
              "$ref": "#/definitions/uint"
            },
            "maxLength": {
              "$ref": "#/definitions/uint"
            },
            "pattern": {
              "type": "string"
            },
            "format": {
              "$ref": "#/definitions/format-"
            },
            "minItems": {
              "$ref": "#/definitions/uint"
            },
            "maxItems": {
              "$ref": "#/definitions/uint"
            },
            "uniqueItems": {
              "type": "boolean"
            },
            "items": {
              "$ref": "#/definitions/jso-items"
            },
            "unit": {
              "type": "string"
            },
            "nullable": {
              "type": "boolean"
            },
```

```
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "const": "object"
              },
              "required": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "properties": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "sdfChoice": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "description": {
                "type": "string"
              },
              "label": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "sdfRequired": {
                "$ref": "#/definitions/pointer-list"
              },
              "const": {
                "$ref": "#/definitions/allowed-types"
              },
              "default": {
                "$ref": "#/definitions/allowed-types"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
```

```
              },
              "exclusiveMinimum": {
                "type": "number"
              },
              "exclusiveMaximum": {
                "type": "number"
              },
              "multipleOf": {
                "type": "number"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              },
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
+           "properties": {
+             "type": {
+               "type": "string"
+             },
```

```
+            "sdfChoice": {
+              "$ref": "#/definitions/sdfData-sdfChoice-properties-"
+            },
+            "description": {
+              "type": "string"
+            },
+            "label": {
+              "type": "string"
+            },
+            "$comment": {
+              "type": "string"
+            },
+            "sdfRef": {
+              "$ref": "#/definitions/sdf-pointer"
+            },
+            "sdfRequired": {
+              "$ref": "#/definitions/pointer-list"
+            },
+            "const": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "default": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "minimum": {
+              "type": "number"
+            },
+            "maximum": {
+              "type": "number"
+            },
+            "exclusiveMinimum": {
+              "type": "number"
+            },
+            "exclusiveMaximum": {
+              "type": "number"
+            },
+            "multipleOf": {
+              "type": "number"
+            },
+            "minLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "pattern": {
+              "type": "string"
+            },
+            "format": {
+              "$ref": "#/definitions/format-"
+            },
+            "minItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "uniqueItems": {
```

```
+               "type": "boolean"
+             },
+             "items": {
+               "$ref": "#/definitions/jso-items"
+             },
+             "unit": {
+               "type": "string"
+             },
+             "nullable": {
+               "type": "boolean"
+             },
+             "sdfType": {
+               "$ref": "#/definitions/sdfType-"
+             },
+             "contentFormat": {
+               "type": "string"
+             }
+           },
+           "additionalProperties": false
+         },
+         {
+           "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
             "properties": {
               "type": {
                 "$ref": "#/definitions/type-"
               },
               "enum": {
                 "type": "array",
                 "items": {
                   "type": "string"
                 },
                 "minItems": 1
               },
               "description": {
                 "type": "string"
               },
               "label": {
                 "type": "string"
               },
               "$comment": {
                 "type": "string"
               },
               "sdfRef": {
                 "$ref": "#/definitions/sdf-pointer"
               },
               "sdfRequired": {
                 "$ref": "#/definitions/pointer-list"
               },
               "const": {
                 "$ref": "#/definitions/allowed-types"
               },
               "default": {
                 "$ref": "#/definitions/allowed-types"
               },
               "minimum": {
```

```
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "exclusiveMinimum": {
                "type": "number"
              },
              "exclusiveMaximum": {
                "type": "number"
              },
              "multipleOf": {
                "type": "number"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              },
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
```

```
            "properties": {
              "type": {
                "type": "string",
                "const": "object"
              },
              "required": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "properties": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "enum": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "description": {
                "type": "string"
              },
              "label": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "sdfRequired": {
                "$ref": "#/definitions/pointer-list"
              },
              "const": {
                "$ref": "#/definitions/allowed-types"
              },
              "default": {
                "$ref": "#/definitions/allowed-types"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "exclusiveMinimum": {
                "type": "number"
              },
              "exclusiveMaximum": {
                "type": "number"
              },
              "multipleOf": {
                "type": "number"
              },
```

```
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              },
              "pattern": {
                "type": "string"
              },
              "format": {
                "$ref": "#/definitions/format-"
              },
              "minItems": {
                "$ref": "#/definitions/uint"
              },
              "maxItems": {
                "$ref": "#/definitions/uint"
              },
              "uniqueItems": {
                "type": "boolean"
              },
              "items": {
                "$ref": "#/definitions/jso-items"
              },
              "unit": {
                "type": "string"
              },
              "nullable": {
                "type": "boolean"
              },
              "sdfType": {
                "$ref": "#/definitions/sdfType-"
              },
              "contentFormat": {
                "type": "string"
              }
            },
            "additionalProperties": false
+         },
+         {
+           "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
+           "properties": {
+             "type": {
+               "type": "string"
+             },
+             "enum": {
+               "type": "array",
+               "items": {
+                 "type": "string"
+               },
+               "minItems": 1
+             },
+             "description": {
+               "type": "string"
+             },
```

```
+            "label": {
+              "type": "string"
+            },
+            "$comment": {
+              "type": "string"
+            },
+            "sdfRef": {
+              "$ref": "#/definitions/sdf-pointer"
+            },
+            "sdfRequired": {
+              "$ref": "#/definitions/pointer-list"
+            },
+            "const": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "default": {
+              "$ref": "#/definitions/allowed-types"
+            },
+            "minimum": {
+              "type": "number"
+            },
+            "maximum": {
+              "type": "number"
+            },
+            "exclusiveMinimum": {
+              "type": "number"
+            },
+            "exclusiveMaximum": {
+              "type": "number"
+            },
+            "multipleOf": {
+              "type": "number"
+            },
+            "minLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxLength": {
+              "$ref": "#/definitions/uint"
+            },
+            "pattern": {
+              "type": "string"
+            },
+            "format": {
+              "$ref": "#/definitions/format-"
+            },
+            "minItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "maxItems": {
+              "$ref": "#/definitions/uint"
+            },
+            "uniqueItems": {
+              "type": "boolean"
+            },
+            "items": {
+              "$ref": "#/definitions/jso-items"
+            },
+            "unit": {
```

```
+               "type": "string"
+             },
+             "nullable": {
+               "type": "boolean"
+             },
+             "sdfType": {
+               "$ref": "#/definitions/sdfType-"
+             },
+             "contentFormat": {
+               "type": "string"
+             }
+           },
+           "additionalProperties": false
         }
       ]
     },
     "allowed-types": {
       "anyOf": [
         {
           "type": "number"
         },
         {
           "type": "string"
         },
         {
           "type": "boolean"
         },
         {
           "type": "null"
         },
         {
           "type": "array",
           "items": {
             "type": "number"
           }
         },
         {
           "type": "array",
           "items": {
             "type": "string"
           }
         },
         {
           "type": "array",
           "items": {
             "type": "boolean"
           }
         },
         {
           "type": "object",
           "additionalProperties": {}
-        }
+        },
+        {}
       ]
     },
     "uint": {
       "type": "integer",
```

```
        "minimum": 0
      },
      "jso-items": {
        "anyOf": [
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "enum": [
                  "number",
                  "string",
                  "boolean",
                  "integer"
                ]
              },
              "sdfChoice": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "description": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "format": {
                "type": "string"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "const": "object"
```

```
              },
              "required": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "properties": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "sdfChoice": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "description": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "format": {
                "type": "string"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
+           "properties": {
+             "type": {
+               "type": "string"
+             },
+             "sdfChoice": {
+               "$ref": "#/definitions/sdfData-sdfChoice-properties-"
+             },
+             "sdfRef": {
+               "$ref": "#/definitions/sdf-pointer"
+             },
+             "description": {
+               "type": "string"
```

```
+             },
+             "$comment": {
+               "type": "string"
+             },
+             "minimum": {
+               "type": "number"
+             },
+             "maximum": {
+               "type": "number"
+             },
+             "format": {
+               "type": "string"
+             },
+             "minLength": {
+               "$ref": "#/definitions/uint"
+             },
+             "maxLength": {
+               "$ref": "#/definitions/uint"
+             }
+           },
+           "additionalProperties": false
+         },
+         {
+           "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "enum": [
                  "number",
                  "string",
                  "boolean",
                  "integer"
                ]
              },
              "enum": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "description": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
```

```
              },
              "format": {
                "type": "string"
              },
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              }
            },
            "additionalProperties": false
          },
          {
            "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
            "properties": {
              "type": {
                "type": "string",
                "const": "object"
              },
              "required": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "properties": {
                "$ref": "#/definitions/sdfData-sdfChoice-properties-"
              },
              "enum": {
                "type": "array",
                "items": {
                  "type": "string"
                },
                "minItems": 1
              },
              "sdfRef": {
                "$ref": "#/definitions/sdf-pointer"
              },
              "description": {
                "type": "string"
              },
              "$comment": {
                "type": "string"
              },
              "minimum": {
                "type": "number"
              },
              "maximum": {
                "type": "number"
              },
              "format": {
                "type": "string"
              },
```

```
              "minLength": {
                "$ref": "#/definitions/uint"
              },
              "maxLength": {
                "$ref": "#/definitions/uint"
              }
            },
            "additionalProperties": false
+         },
+         {
+           "type": "object",
+           "patternProperties": {
+             "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+           },
+           "properties": {
+             "type": {
+               "type": "string"
+             },
+             "enum": {
+               "type": "array",
+               "items": {
+                 "type": "string"
+               },
+               "minItems": 1
+             },
+             "sdfRef": {
+               "$ref": "#/definitions/sdf-pointer"
+             },
+             "description": {
+               "type": "string"
+             },
+             "$comment": {
+               "type": "string"
+             },
+             "minimum": {
+               "type": "number"
+             },
+             "maximum": {
+               "type": "number"
+             },
+             "format": {
+               "type": "string"
+             },
+             "minLength": {
+               "$ref": "#/definitions/uint"
+             },
+             "maxLength": {
+               "$ref": "#/definitions/uint"
+             }
+           },
+           "additionalProperties": false
          }
        ]
      },
+     "sdftype-name": {
+       "type": "string",
+       "pattern": "^[a-z][\\-a-z0-9]*$"
+     },
```

```
      "actionqualities": {
        "type": "object",
        "properties": {
          "description": {
            "type": "string"
          },
          "label": {
            "type": "string"
          },
          "$comment": {
            "type": "string"
          },
          "sdfRef": {
            "$ref": "#/definitions/sdf-pointer"
          },
          "sdfRequired": {
            "$ref": "#/definitions/pointer-list"
          },
          "sdfInputData": {
            "$ref": "#/definitions/parameter-list"
          },
          "sdfOutputData": {
            "$ref": "#/definitions/parameter-list"
          },
          "sdfData": {
            "$ref": "#/definitions/sdfData-sdfChoice-properties-"
          }
        },
+       "patternProperties": {
+         "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+       },
        "additionalProperties": false
      },
      "parameter-list": {
        "$ref": "#/definitions/dataqualities"
      },
      "eventqualities": {
        "type": "object",
        "properties": {
          "description": {
            "type": "string"
          },
          "label": {
            "type": "string"
          },
          "$comment": {
            "type": "string"
          },
          "sdfRef": {
            "$ref": "#/definitions/sdf-pointer"
          },
          "sdfRequired": {
            "$ref": "#/definitions/pointer-list"
          },
          "sdfOutputData": {
            "$ref": "#/definitions/parameter-list"
          },
          "sdfData": {
```

```
            "$ref": "#/definitions/sdfData-sdfChoice-properties-"
          }
        },
+       "patternProperties": {
+         "^(?:[a-z][a-z0-9]*:)?[a-z$][A-Za-z$0-9]*$": {}
+       },
        "additionalProperties": false
      },
      "format-": {
-       "type": "string",
-       "enum": [
-         "date-time",
-         "date",
-         "time",
-         "uri",
-         "uri-reference",
-         "uuid"
+       "anyOf": [
+         {
+           "type": "string",
+           "const": "date-time"
+         },
+         {
+           "type": "string",
+           "const": "date"
+         },
+         {
+           "type": "string",
+           "const": "time"
+         },
+         {
+           "type": "string",
+           "const": "uri"
+         },
+         {
+           "type": "string",
+           "const": "uri-reference"
+         },
+         {
+           "type": "string",
+           "const": "uuid"
+         },
+         {
+           "type": "string"
+         }
+       ]
+     },
+     "sdfType-": {
+       "anyOf": [
+         {
+           "type": "string",
+           "const": "byte-string"
+         },
+         {
+           "type": "string",
+           "const": "unix-time"
+         },
+         {
```

```
+          "$ref": "#/definitions/sdftype-name"
+        }
       ]
     },
     "sdfData-sdfChoice-properties-": {
       "type": "object",
       "additionalProperties": {
         "$ref": "#/definitions/dataqualities"
       }
     },
     "type-": {
       "type": "string",
       "enum": [
         "number",
         "string",
         "boolean",
         "integer",
         "array"
       ]
     },
-    "sdfEvent-": {
+    "sdfProperty-": {
       "type": "object",
       "additionalProperties": {
-        "$ref": "#/definitions/eventqualities"
+        "$ref": "#/definitions/propertyqualities"
       }
     },
     "sdfAction-": {
       "type": "object",
       "additionalProperties": {
         "$ref": "#/definitions/actionqualities"
       }
     },
-    "sdfProperty-": {
+    "sdfEvent-": {
       "type": "object",
       "additionalProperties": {
-        "$ref": "#/definitions/propertyqualities"
+        "$ref": "#/definitions/eventqualities"
       }
-    },
-    "sdfType-": {
-      "type": "string",
-      "enum": [
-        "byte-string",
-        "unix-time"
-      ]
     }
   }
 }
```

Appendix C. Data Qualities Inspired by json-schema.org

This appendix is normative.

Data qualities define data used in SDF affordances at an information model level. A popular way to describe JSON data at a data model level is proposed by a number of drafts on json-schema.org (which collectively are abbreviated JSO here); for reference to a popular version, this appendix points to [JSO7] and [JSO7V]. As the vocabulary used by JSO is familiar to many JSON modelers, the present specification borrows some of the terms and ports their semantics to the information model level needed for SDF.

The main data quality imported is the "`type`". In SDF, this can take one of six (text string) values, which are discussed in the following subsections (note that the JSO type "`null`" is not supported as a value of this data quality in SDF).

The additional quality "`const`" restricts the data to one specific value (given as the value of the `const` quality).

Similarly, the additional quality "`default`" provides data that can be used in the absence of the data (given as the value of the `default` quality); this is mainly documentary and not very well-defined for SDF as no process is defined that would add default values to an instance of some interaction data.

Other qualities that are inspired by JSO are "`$comment`" and "`description`", both of which are also available in the information block.

C.1. type "`number`", type "`integer`"

The types "`number`" and "`integer`" are associated with floating point and integer numbers, as they are available in JSON. A type value of `integer` means that only integer values of JSON numbers can be used (note that `10.0` is an integer value, even if it is in a notation that would also allow non-zero decimal fractions).

The additional data qualities "`minimum`", "`maximum`", "`exclusiveMinimum`", and "`exclusiveMaximum`" provide number values that serve as inclusive/exclusive lower/upper bounds for the number. (Note that the Boolean form of "`exclusiveMinimum`"/"`exclusiveMaximum`" found in earlier JSO drafts [JSO4V] is not used.)

The data quality "`multipleOf`" gives a positive number that constrains the data value to be an integer multiple of the number given. (Type "`integer`" can also be expressed as a "`multipleOf`" quality of value 1, unless another "`multipleOf`" quality is present.)

C.2. type "`string`"

The type "`string`" is associated with Unicode text string values, as they can be represented in JSON.

The length (as measured in characters, specifically Unicode scalar values) can be constrained by the additional data qualities "`minLength`" and "`maxLength`", which are inclusive bounds.

(More specifically, Unicode text strings as defined in this specification are sequences of Unicode scalar values, the number of which is taken as the length of such a text string.

The data quality "pattern" takes a string value that is interpreted as an [ECMA-262] regular expression in Unicode mode that constrains the string (note that these are not anchored by default, so unless ^ and $ anchors are employed, ECMA-262 regular expressions match any string that *contains* a match). The JSO proposals acknowledge that regular expression support is rather diverse in various platforms, so the suggestion is to limit them to:

- characters;
- character classes in square brackets, including ranges; their complements;
- simple quantifiers *, +, ?, and range quantifiers {n}, {n,m}, and {n,};
- grouping parentheses;
- the choice operator |;
- and anchors (beginning-of-input ^ and end-of-input $).

Note that this subset is somewhat similar to the subset introduced by I-Regexps [RFC9485], which are anchored regular expressions and include certain backslash escapes for characters and character classes.

The additional data quality "format" can take one of the following values. Note that, at an information model level, the presence of this data quality changes the type from being a simple text string to the abstract meaning of the format given (i.e., the format "date-time" is less about the specific syntax employed in [RFC3339] than about the usage as an absolute point in civil time).

- "date-time", "date", "time": A date-time, full-date, or full-time as defined in [RFC3339], respectively.
- "uri", "uri-reference": A URI or URI Reference as defined in [STD66], respectively.
- "uuid": A Universally Unique Identifier (UUID) as defined in [RFC9562]).

C.3. type "boolean"

The type "boolean" can take the values "true" or "false".

C.4. type "array"

The type "array" is associated with arrays, as they are available in JSON.

The additional quality "items" gives the type that each of the elements of the array must match.

The number of elements in the array can be constrained by the additional data qualities "minItems" and "maxItems", which are inclusive bounds.

The additional data quality "uniqueItems" gives a Boolean value that, if true, requires the elements to be all different.

C.5. type "object"

The type "object" is associated with maps, from strings to values, as they are available in JSON.

The additional quality "`properties`" is a map the entries of which describe entries in the specified JSON map: the key gives an allowable map key for the specified JSON map and the value is a map with a named set of data qualities giving the type for the corresponding value in the specified JSON map.

All entries specified in this way are optional unless they are listed in the value of the additional quality "`required`", which is an array of string values that give the key names of required entries.

Note that the term "properties" as an additional quality for defining map entries is unrelated to sdfProperty.

For example, to include information about the type of the event in the "overTemperatureEvent" of Figure 4, the sdfOutputData there could be defined as follows:

```
    "sdfOutputData": {
      "type": "object",
      "properties": {
        "alarmType": {
          "sdfRef": "cap:#/sdfData/alarmTypes/quantityAlarms",
          "const": "OverTemperatureAlarm"
        },
        "temperature": {
"sdfRef": "#/sdfObject/temperatureWithAlarm/sdfData/temperatureData"
        }
      }
    }
```

Figure 6: Using Object Type with sdfOutputData

C.6. Implementation Notes

JSO-based keywords are also used in the specification techniques of a number of ecosystems, but some adjustments may be required.

For instance, [OCF] is based on Swagger 2.0, which appears to be based on "draft-4" [JSO4] [JSO4V] (also called draft-5, but semantically intended to be equivalent to draft-4). The "`exclusiveMinimum`" and "`exclusiveMaximum`" keywords use the Boolean form there, so on import to SDF, their values have to be replaced by the values of the respective "`minimum`"/"`maximum`" keyword, which are then removed; the reverse transformation applies on export.

Appendix D. Composition Examples

This informative appendix contains two examples illustrating different composition approaches using the `sdfThing` quality.

D.1. Outlet Strip Example

```
{
  "sdfThing": {
    "outlet-strip": {
      "label": "Outlet strip",
      "description": "Contains a number of Sockets",
      "sdfObject": {
        "socket": {
          "description": "An array of sockets in the outlet strip",
          "minItems": 2,
          "maxItems": 10
        }
      }
    }
  }
}
```

Figure 7: Outlet Strip Example

D.2. Refrigerator-Freezer Example

```
{
  "sdfThing": {
    "refrigerator-freezer": {
      "description": "A refrigerator combined with a freezer",
      "sdfProperty": {
        "status": {
          "type": "boolean",
          "description":
"Indicates if the refrigerator-freezer is powered"
        }
      },
      "sdfObject": {
        "refrigerator": {
          "description": "A refrigerator compartment",
          "sdfProperty": {
            "temperature": {
              "sdfRef": "#/sdfProperty/temperature",
              "maximum": 8
            }
          }
        },
        "freezer": {
          "label": "A freezer compartment",
          "sdfProperty": {
            "temperature": {
              "sdfRef": "#/sdfProperty/temperature",
              "maximum": -6
            }
          }
        }
      }
    }
  },
  "sdfProperty": {
    "temperature": {
      "description": "The temperature for this compartment",
      "type": "number",
      "unit": "Cel"
    }
  }
}
```

Figure 8: Refrigerator-Freezer Example

Appendix E. Some Changes from Earlier Draft Versions of this Specification

This appendix is informative.

The present document provides the base SDF definition. Previous revisions of SDF, as defined in earlier drafts of this specification, have been in use for several years; both significant collections of older SDF models and older SDF conversion tools are available today. This appendix provides a brief checklist that can aid in upgrading these to the standard.

- The quality `unit` was previously called `units`.
- `sdfType` was developed out of a concept previously called `subtype`.
- `sdfChoice` is the preferred way to represent JSO `enum` (only a limited form of which is retained) and also the way to represent JSO `anyOf`.
- The length of text strings (as used with `minLength`/`maxLength` constraints) was previously defined in bytes. It now is defined as the number of characters (Unicode scalar values, to be exact); a length in bytes is not meaningful unless bound to a specific encoding, which might differ from UTF-8 in some ecosystem mappings and protocol bindings.

List of Figures

List of Tables

Acknowledgements

This specification is based on work by the One Data Model group.

Contributors

Jan Romann
Universität Bremen
Germany
Email: jan.romann@uni-bremen.de

Wouter van der Beek
Cascoda Ltd.
Threefield House
Threefield Lane
Southampton
United Kingdom
Email: w.vanderbeek@cascoda.com

Authors' Addresses

Michael Koster (EDITOR)
KTC Control AB
29415 Alderpoint Road
Blocksburg, CA 95514
United States of America
Phone: +1-707-502-5136
Email: michaeljohnkoster@gmail.com

Carsten Bormann (EDITOR)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Ari Keränen
Ericsson
FI-02420 Jorvas
Finland
Email: ari.keranen@ericsson.com