Workgroup: T2TRG
Internet-Draft: draft-ietf-asdf-sdf-01
Published: 15 November 2020
Intended Status: Informational
Expires: 19 May 2021
Authors: M. Koster, Ed. C. Bormann, Ed.
SmartThings Universität Bremen TZI

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 in the Internet of Things. It was created as a common language for use in the development of the One Data Model liaison organization (OneDM) definitions. Tools convert this format to database formats and other serializations as needed.

An SDF specification describes definitions of SDF Objects and their associated interactions (Events, Actions, Properties), as well as the Data types for the information exchanged in those interactions.

A JSON format representation of SDF 1.0 was defined in the previous (-00) version of this document. SDF 1.1 is expected to be defined in a future version; the present document represents a draft on the way from 1.0 to 1.1. Hence, this is not an implementation draft.

Contributing

Recent versions of this document are available at its GitHub repository https://github.com/ietf-wg-asdf/SDF -- this also provides an issue tracker as well as a way to supply "pull requests".

General discussion of this SDF Internet-Draft happens on the mailing list of the IETF ASDF Working Group, asdf@ietf.org (subscribe at https://www.ietf.org/mailman/listinfo/asdf).

The IETF Note Well applies (https://www.ietf.org/about/note-well/).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 May 2021.

Copyright Notice

Table of Contents

1. Introduction

The Semantic Definition Format (SDF) is a format for domain experts to use in the creation and maintenance of data and interaction models in the Internet of Things. It was created as a common language for use in the development of the One Data Model liaison organization (OneDM) definitions. Tools convert this format to database formats and other serializations as needed.

An SDF specification describes definitions of SDF Objects and their associated interactions (Events, Actions, Properties), as well as the Data types for the information exchanged in those interactions.

A JSON format representation of SDF 1.0 was defined in the previous (-00) version of this document. SDF 1.1 is expected to be defined in a future version; the present document represents a draft on the way from 1.0 to 1.1. Hence, this is not an implementation draft.

1.1. Terminology and Conventions

Thing: A physical device that is also made available in the Internet of Things. The term is used here for Things that are notable for their interaction with the physical world beyond interaction with humans; a temperature sensor or a light might be a Thing, but a router that employs both temperature sensors and indicator lights might exhibit less Thingness, as the effects of its functioning are mostly on the digital side.

Affordance:
An element of an interface offered for interaction, defining its possible uses or making clear how it can or should be used. The term is used here for the digital interfaces of a Thing only; it might also have physical affordances such as buttons, dials, and displays.

Quality: A metadata item in a definition or declaration which says something about that definition or declaration. A quality is represented in SDF as an entry in a JSON map (object) that serves as a definition or declaration.

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

Block: One or more entries in a JSON map that is part of an SDF specification; these entries together serve a specific function.

Group: An entry in the main SDF map and in certain nested definitions that has a Class Name Keyword as its key and a map of definition entries (Definition Group) as a value.

Class Name Keyword: One of sdfThing, sdfProduct, 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.

Property: An affordance that can potentially be used to read, write, and/or observe state on an Object. (Note that Entries are often called properties in other environments; in this document, the term Property is specifically reserved for affordances, even if the map key "properties" might be imported from a data definition language with the other semantics.)

Action: An affordance that can potentially be used to perform a named operation on an Object.

Event: An affordance that can potentially be used to obtain information about what happened to an Object.

Object: A grouping of Property, Action, and Event definitions; the main "atom" of reusable semantics for model construction. (Note that JSON maps are often called JSON objects due to JSON's JavaScript heritage; in this document, the term Object is specifically reserved for the above grouping, even if the type name "object" might be imported from a data definition language with the other semantics.)

Element:
A part or an aspect of something abstract; used here in its usual English definition. (Occasionally, also used specifically for the elements of JSON arrays.)

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.

Declaration: A reference to and a use of a definition within an enclosing definition, intended to create component instances within that enclosing definition. Every declaration can also be used as a definition for reference in a different place.

Protocol Binding: A companion document to an SDF specification that defines how to map the abstract concepts in the specification into the protocols in use in a specific ecosystem. Might supply URL components, numeric IDs, and similar details.

Conventions:

*The singular form is chosen as the preferred one for the keywords defined here.

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

We start with an example for the SDF definition of a simple Object called "Switch" (Figure 1).

```
{
    "info": {
        "title": "Example file for OneDM 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 o
                    "type": "boolean"
                }
            },
            "sdfAction": {
                "on": {
                    "description": "Turn the switch on; equivalent to se
                },
                "off": {
                    "description": "Turn the switch off; equivalent to s
                },
                "toggle": {
                    "description": "Toggle the switch; equivalent to set
                }
            }
        }
    }
}
```

Figure 1: A simple example of an SDF definition file

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 first retrieving the current value and then applying/setting the inverted value.

The sdfObject group lists the affordances of instances of this object. The sdfProperty group lists the property affordances described by the model; these represent various perspectives on the

state of the object. 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 [I-D.irtf-t2trg-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.)

2.2. Elements of an SDF model

The SDF language uses seven predefined Class Name Keywords for modeling connected Things, six of which are illustrated in Figure 2 (the seventh class sdfProduct is exactly like sdfThing).

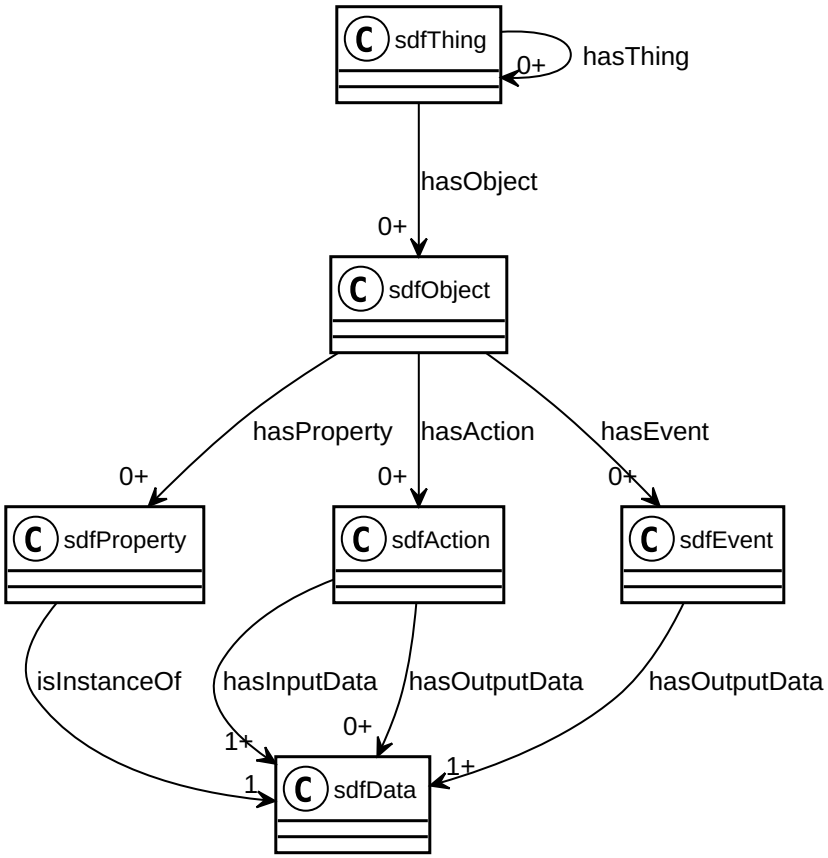

Figure 2: Main classes used in SDF models

The seven main Class Name Keywords are discussed below.

2.2.1. sdfObject

Objects, the items listed in an sdfObject group, are the main "atom" of reusable semantics for model construction. It aligns in scope with common definition items from many IoT modeling systems, for example ZigBee Clusters [ZCL], OMA SpecWorks LwM2M Objects [OMA], and OCF Resource Types [OCF].

An sdfObject 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 may different kinds of Things that require on/off control.

2.2.2. sdfProperty

sdfProperty is used to model elements of state within sdfObject instances.

An instance of 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 definitions in sdfData groups, however, they actually also declare a Property with the given qualities to be potentially present in the containing Object. (Qualities beyond those of sdfData definitions could be defined for sdfProperty declarations but currently aren't; this means that even Property qualities such as readable and writable can be associated with definitions in sdfData groups, as well.)

For definitions in sdfProperty and sdfData, SDF provides qualities that can constrain the structure and values of data allowed in an instance of these data, as well as qualities that associate semantics to these data, for engineering units and unit scaling information.

For the data definition within sdfProperty or sdfData, SDF borrows a number of elements proposed for the drafts 4 and 7 of the json-schema.org "JSON Schema" format [I-D.handrews-json-schema-

validation], enhanced by qualities that are specific to SDF. For the current version of SDF, data are constrained to be of simple types (number, string, Boolean), JSON maps composed of named data ("objects"), and arrays of these types. Syntax extension points are provided that can be used to provide richer types in future versions 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, model affordances that, when triggered, have more effect than just reading, updating, or observing Thing state, often resulting 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 modeled as a single item of input data and output data, each. (Where multiple parameters need to be modeled, an "object" type can be used to combine these parameters into one.) 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 sdfPoperty; 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.

Actions may have (or lack) qualities of idempotency and side-effect safety.

The current version of 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 can model affordances that inform about "happenings" associated with an instance of an Object; 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.

The current version of 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 invoking the action, are expected to be part of the protocol binding.

2.2.5. sdfData

Definitions in sdfData groups 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.

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 groups enables definition of models for complex devices that will use one or more sdfObject definitions.

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

2.2.7. sdfProduct

sdfThing has a derived class sdfProduct, which can be used to indicate a top level inventory item with a Stock-Keeping Unit (SKU) identifier and other particular metadata. Structurally, there is no difference between definitions in either group; semantically, a definition in an sdfProduct group is intended to describe a class of complete Things.

3. SDF structure

SDF definitions are contained in SDF files. One or more SDF files can work together to provide the definitions and declarations that are the payload of the SDF format.

A SDF definition file contains a single JSON map (JSON object). This object has three sections: the information block, the namespaces section, and the definitions section.

3.1. Information block

The information block contains generic meta data for the file itself and all included definitions.

The keyword (map key) that defines an information block is "info". Its value is a JSON map in turn, with a set of entries that represent qualities that apply to the included definition.

Qualities of the information block are shown in Table 1.

Quality	Type	Required	Description
title	string	yes	A short summary to be displayed in search results, etc.
version	string	yes	The incremental version of the definition, format TBD
copyright	string	yes	Link to text or embedded text containing a copyright notice
license	string	yes	Link to text or embedded text containing license terms

Table 1: Qualities of the Information Block

While the format of the version string is marked as TBD, it is intended 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 an [RFC3339] date-time 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).

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. (For models to be handled by the One Data Model liaison group, this will typically be "BSD-3-Clause".)

3.2. Namespaces section

The namespaces section contains the namespace map and the defaultNamespace setting.

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 definition file.

Quality	Type	Required	Description
namespace	map	no	Defines short names mapped to namespace URIs, to be used as identifier prefixes
defaultNamespace	string	no	Identifies one of the prefixes in the namespace map to be used as a default in resolving identifiers

Table 2: Namespaces Section

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",
```

If no defaultNamespace setting is given, the SDF definition file does not contribute to a global namespace. As the defaultNamespace is set by giving 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 section

The Definitions section contains one or more groups, each identified by a Class Name Keyword (there can only be one group per keyword; the actual grouping is just a shortcut and does not carry any specific semantics). The value of each group is a JSON map (object), 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, we speak of the map entries as "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 current definition block.

A definition may in turn 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 Object definition is given in Figure 3:

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

Figure 3: Example Object definition

This example defines an Object "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.

Some of the definitions are also declarations: the definition of the entry "bar" in the property "foo" means that each instance of a "foo" can have zero or one instance of a "bar". Entries within sdfProperty, sdfAction, and sdfEvent, within sdfObject entries, are declarations. Similarly, entries within an sdfThing describe instances of sdfObject (or nested sdfThing) that form part of instances of the Thing.

4. Names and namespaces

SDF definition files may contribute to a global namespace, and may reference elements from that global namespace. (An SDF definition file 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 versions 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.)

The absolute URI of a global name should be a URI as per Section 3 of [RFC3986], with a scheme of "https" and a path (hier-part in [RFC3986]). For the present version of this specification, the query part should not be used (it might be used in later versions).

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 definition file is constructed from a JSON pointer that selects the element defined for this name in the SDF definition file.

The absolute URI part is a copy of the default namespace, i.e., the default namespace is always the target namespace for a name for which a definition is contributed. When emphasizing that name definitions are contributed to the default namespace, we therefore also call it the "target namespace" of the SDF definition file.

E.g., 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

Note the #, which separates the absolute-URI part (Section 4.3 of [RFC3986]) from the fragment identifier part.

4.3. Referencing global names

A name reference takes the form of the production curie in [W3C.NOTE-curie-20101216] (note that this excludes the production

safe-curie), but also limiting the IRIs involved in that production to URIs as per [RFC3986] and the prefixes to ASCII characters [RFC0020].

A name that is contributed by the current SDF definition file can be referenced by a Same-Document Reference as per section 4.4 of [RFC3986]. As there is little point in referencing the entire SDF definition file, 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 definition file that does not set a default namespace.

Name references that point outside the current SDF definition file need to contain curie prefixes. These then reference namespace declarations in the namespaces section.

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 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 all of the qualities 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 [I-D.handrews-json-schema-validation].)

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.

4.5. sdfRequired

The value of "sdfRequired" is an array of name references, each pointing to one declaration the instantiation of which is declared mandatory.

4.5.1. Optionality using the keyword "sdfRequired"

The keyword "sdfRequired" is provided to apply a constraint that defines for which declarations corresponding data are mandatory in an instance conforming the current definition.

The value of "sdfRequired" is an array of JSON pointers, each indicating one declaration that is mandatory to be represented.

The example in Figure 4 shows two required elements in the sdfObject definition for "temperatureWithAlarm", the sdfProperty "temperatureData", and the sdfEvent "overTemperatureEvent". The example also shows the use of JSON pointer with "sdfRef" to use a pre-existing definition in this definition, for the "alarmType" data (sdfOutputData) produced by the sdfEvent "overTemperatureEvent".

```
{
  "sdfObject": {
    "temperatureWithAlarm": {
      "sdfRequired": [
        "#/sdfObject/temperatureWithAlarm/sdfData/temperatureData",
        "#/sdfObject/temperatureWithAlarm/sdfEvent/overTemperatureEvent"
      ],
      "sdfData":{
        "temperatureData": {
          "type": "number"
        }
      },
      "sdfEvent": {
        "overTemperatureEvent": {
          "sdfOutputData": {
            "type": "object",
            "properties": {
              "alarmType": {
                "sdfRef": "cap:/sdfData/alarmTypes/quantityAlarms",
                "const": "OverTemperatureAlarm"
              },
              "temperature": {
                "sdfRef": "#/sdfObject/temperatureWithAlarm/sdfData/temp
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 4: Using sdfRequired

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 label is required for an application and no label is given in the SDF model, the last part (reference-token, Section 3 of [RFC6901]) of the JSON pointer to the definition can be used.

Quality	Type	Description
description	text	long text (no constraints)
label	text	short text (no constraints)
$comment	text	source code comments only, no semantics
sdfRef		(see Section 4.4)

Quality	Type	Description
	sdf-pointer	
sdfRequired	pointer-list	(see Section 4.5, applies to qualities of properties, of data)

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).

Table 4 lists data qualities borrowed from [I-D.handrews-json-schema-validation]; the intention is that these qualities retain their semantics from the versions of the json-schema.org proposal they were imported from. A description that starts with a parenthesized term means the quality is only applicable when type has the value of the term.

Table 5 lists data qualities defined specifically for the present specification.

The term "allowed types" stands for primitive JSON types, JSON maps ("objects")" as well as homogeneous arrays of numbers, text, Booleans, or maps. (This list might be extended in a future version of SDF.) An "allowed value" is a value allowed for one of these types.

Quality	Type	Description
type	"number" / "string" / "boolean" / "integer" / "array" / "object"	JSON data type (note 1)
enum	array of allowed values	enumeration constraint
const	allowed value	specifies a constant value for a data item or property
default	allowed value	specifies the default value for initialization
minimum	number	(number) lower limit of value
maximum	number	(number) upper limit of value
exclusiveMinimum	number or boolean (jso draft 7/4)	(number) lower limit of value
exclusiveMaximum	number or boolean (jso draft 7/4)	(number) lower limit of value
multipleOf	number	(number) resolution of the number [NEEDED?]

Quality	Type	Description
minLength	integer	(string) shortest length string in octets
maxLength	integer	(string) longest length string in octets
pattern	string	(string) regular expression to constrain a string pattern
format	"date-time" / "date" / "time" / "uri" / "uri-reference" / "uuid"	(string) JSON Schema formats as per [I-D.handrews-json-schema-validation], Section 7.3
minItems	number	(array) Minimum number of items in array
maxItems	number	(array) Maximum number of items in array
uniqueItems	boolean	(array) if true, requires items to be all different
items	(subset of common/ data qualities; see Appendix A	(array) constraints on array items
required	array of strings	(object) names of properties (note 2) that are required in the JSON map ("object")
properties	named set of data qualities	(object) entries allowed for the JSON map ("object")

Table 4: Qualities of sdfProperty and sdfData borrowed from json-schema.org

(1) A type value of integer means that only integral values of JSON numbers can be used.

(2) Note that the term "properties" as used for map entries in [I-D.handrews-json-schema-validation] is unrelated to sdfProperty.

Quality	Type	Description	Default
(common)		Section 4.6	
unit	string	SenML unit name as per [IANA.senml], subregistry SenML Units (note 3)	N/A
scaleMinimum	number	lower limit of value in units given by unit	N/A
scaleMaximum	number	upper limit of value in units given by unit	N/A
readable	boolean	Reads are allowed	true
writable	boolean	Writes are allowed	true

Quality	Type	Description	Default
observable	boolean	flag to indicate asynchronous notification is available	true
nullable	boolean	indicates a null value is available for this type	true
contentFormat	string	content type (IANA media type string plus parameters), encoding	N/A
subtype	"byte-string" / "unix-time"	subtype enumeration	N/A

Table 5: SDF-defined Qualities of sdfProperty and sdfData

(3) note that the quality unit was called units in SDF 1.0.

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 Object definitions. Object definitions may contain or include definitions of Properties, Actions, Events declared for the object, as well as data types (sdfData group) to be used in this or other Objects.

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 object
sdfAction	action	zero or more named action definitions for this object
sdfEvent	event	zero or more named event definitions for this object
sdfData	named-sdq	zero or more named data type definitions that might be used in the above

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.

5.3. sdfAction

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

Actions are used to model commands and methods which are invoked. Actions have parameter data that are supplied upon invocation.

The qualities of an Action definition include the common qualities, additional qualities are shown in Table 7.

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 7: 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 which are communicated upon the occurrence of the event.

The qualities of sdfEvent include the common qualities, additional qualities are shown in Table 8.

Quality	Type	Description
(common)		Section 4.6
sdfOutputData	map	data qualities of the output data for an Event

Quality	Type	Description
sdfData	named-sdq	zero or more named data type definitions that might be used in the above

Table 8: 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. It is not itself a declaration, i.e., 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 Objects.

* The ability to compose a reusable definition block from Objects, for example a single plug unit of an outlet strip with on/off control, energy monitor, and optional dimmer objects, while retaining the atomicity of the individual objects.

* The ability to compose Objects and other definition blocks into a higher level thing that represents a product, while retaining the atomicity of objects.

* The ability to enrich and refine a base definition to have product-specific qualities and quality values, e.g. unit, range, and scale settings.

* The ability to reference items in one part of a complex definition from another part of the same definition, for example to summarize 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 definition files that are present in 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. There is expected to be an SDF namespace for common SDF definitions used in OneDM.

The structure of a path in a namespace is defined by the JSON Pointers to the definitions in the files in that namespace. For example, if there is a file defining an object "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 (could be in the same file or another file) 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 re-use 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 which uses the defined qualities of some existing definition. This pattern will use 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. Optionally, new qualities may be added and values of optional qualities and quality values may be defined.

ISSUE: Do we want to enable qualities from the source definition to be overridden in future versions? The above only says "added". (Yes, we do want to enable overriding, but need to warn specifiers not to use this in a way that contradicts the referenced semantics.)

```
"sdfData":
  "length" : {
    "type": "number",
    "minimum": 0,
    "unit": "m"
    "description": "There can be no negative lengths."
  }
...
  "cable-length" : {
    "sdfRef": "#/sdfData/length"
    "minimum": 0.05,
    "description": "Cables must be at least 5 cm."
  }
```

6.3. sdfThing

An sdfThing is a set of declarations and qualities that may be part of a more complex model. For example, the object declarations that make up the definition of a single socket of an outlet strip could be encapsulated in an sdfThing, and the socket-thing itself could be used in a declaration in the sdfThing definition for the outlet strip.

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.

The qualities of sdfThing are shown in Table 9.

Quality	Type	Description
(common)		Section 4.6
sdfThing	thing	
sdfObject	object	

Table 9: Qualities of sdfThing and sdfProduct

6.4. sdfProduct

An sdfProduct provides the level of abstraction for representing a unique product or a profile for a standardized type of product, for example a "device type" definition with required minimum functionality.

Products may be composed of Objects and Things at the high level, and may include their own definitions of Properties, Actions, and

Events that can be used to extend or complete the included Object definitions.

Product definitions may set optional defaults and constant values for specific use cases, for example units, range, and scale settings for properties, or available parameters for Actions.

The qualities of sdfProduct are the same as for sdfThing and are shown in Table 9.

7. References

7.1. Normative References

[I-D.handrews-json-schema-validation] Wright, A., Andrews, H., and B. Hutton, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", Work in Progress, Internet-Draft, draft-handrews-json-schema-validation-02, 17 September 2019, <http://www.ietf.org/internet-drafts/draft-handrews-json-schema-validation-02.txt>.

[I-D.ietf-cbor-cddl-control] Bormann, C., "Additional Control Operators for CDDL", Work in Progress, Internet-Draft, draft-ietf-cbor-cddl-control-00, 29 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-cbor-cddl-control-00.txt>.

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

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

[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>.

[RFC3986] 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>.

[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>.

[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>.

[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>.

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

[W3C.NOTE-curie-20101216] Birbeck, M. and S. McCarron, "CURIE Syntax 1.0", World Wide Web Consortium NOTE NOTE-curie-20101216, 16 December 2010, <https://www.w3.org/TR/2010/NOTE-curie-20101216>.

7.2. Informative References

[I-D.irtf-t2trg-rest-iot] Keranen, A., Kovatsch, M., and K. Hartke, "RESTful Design for Internet of Things Systems", Work in Progress, Internet-Draft, draft-irtf-t2trg-rest-iot-06, 11 May 2020, <http://www.ietf.org/internet-drafts/draft-irtf-t2trg-rest-iot-06.txt>.

[OCF] "OCF Resource Type Specification", <https://openconnectivity.org/specs/OCF_Resource_Type_Specification.pdf>.

[OMA] "OMA LightweightM2M (LwM2M) Object and Resource Registry", <http://www.openmobilealliance.org/wp/omna/lwm2m/lwm2mregistry.html>.

[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>.

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

Appendix A. Formal Syntax of SDF

This appendix describes the syntax of SDF using CDDL [RFC8610]. Note that this appendix was derived from Ari Keranen's "alt-schema" and Michael Koster's "schema", with a view of covering the syntax that is currently in use at the One Data Model playground repository.

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 [I-D.ietf-cbor-cddl-control]. A feature named "1.0" is used to indicate parts of the syntax being deprecated towards SDF 1.1, and a feature named "1.1" is used to indicate new syntax intended for SDF 1.1. Features whose names end in "-ext" indicate extension points for further evolution.

```
start = sdf-syntax

sdf-syntax = {
 info: sdfinfo                  ; don't *require* this in flexible synta
 ? namespace: named<text>
 ? defaultNamespace: text
 ? sdfThing: named<thingqualities>       ; Thing is a composition of obj
 ? sdfProduct: named<productqualities>   ; Product is a composition of t
 ? sdfObject: named<objectqualities>     ; Object is a set of Properties
 ? sdfProperty: named<propertyqualities> ; Property represents the state
 ? sdfAction: named<actionqualities>     ; Action is a directive to invo
 ? sdfEvent: named<eventqualities>       ; Event represents an occurence
 ? sdfData: named<dataqualities>         ; Data represents a piece of in
 EXTENSION-POINT<"top-ext">
}

sdfinfo = {
 title: text
 version: text
 copyright: text
 license: text
 EXTENSION-POINT<"info-ext">
}

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

EXTENSION-POINT<f> = ( * (text .feature f) => any ) ; only used in frame

sdf-pointer = text ; .regexp curie-regexp -- TO DO!
pointer-list = [* sdf-pointer]  ; ISSUE: no point in having an empty lis

commonqualities = (
 ? description: text            ; long text (no constraints)
 ? label: text                  ; short text (no constraints); default t
 ? $comment: text               ; source code comments only, no semantic
 ? sdfRef: sdf-pointer
 ? sdfRequired: pointer-list    ; applies to qualities of properties, of
)

; for building hierarchy
thingqualities = {
 commonqualities,
 ? sdfObject: named<objectqualities>
 ? sdfThing: named<thingqualities>
 EXTENSION-POINT<"thing-ext">
}

productqualities = thingqualities ; ISSUE: get rid of sdfProduct?
```

```
; for single objects
objectqualities = {
 commonqualities,
 ? sdfProperty: named<propertyqualities>
 ? sdfAction: named<actionqualities>
 ? sdfEvent: named<eventqualities>
 ? sdfData: named<dataqualities>
 EXTENSION-POINT<"object-ext">
}

propertyqualities = dataqualities ; the definitions in sdfData are decla

parameter-list =
  pointer-list .feature (["1.0", "pointerlist-as-parameter"]) /
  dataqualities .feature (["1.1", "dataqualities-as-parameter"])

actionqualities = {
 commonqualities,
 ? sdfInputData: parameter-list   ; sdfRequiredInputData applies here (a
 ? sdfRequiredInputData: pointer-list
 ? sdfOutputData: parameter-list  ; sdfRequired applies here
 ? sdfData: named<dataqualities>         ; zero or more named data type
 EXTENSION-POINT<"action-ext">
}

eventqualities = {
 commonqualities
 ? sdfOutputData: parameter-list  ; sdfRequired applies here
 ? sdfData: named<dataqualities>         ; zero or more named data type
 EXTENSION-POINT<"event-ext">
}

dataqualities = {               ; also propertyqualities
 commonqualities,
 jsonschema,
 ? ("units" .feature "1.0") => text
 ? ("unit" .feature "1.1") => text
 ? scaleMinimum: number
 ? scaleMaximum: number
 ? observable: bool
 ? readable: bool
 ? writable: bool
 ? nullable: bool
 ? subtype: "byte-string" / "unix-time"
           / (text .feature "subtype-ext")                      ; EXTEN
 ? contentFormat: text
 EXTENSION-POINT<"data-ext">
}
```

```
allowed-types = number / text / bool / null
              / [* number] / [* text] / [* bool]
              / {* text => any}
              / (any .feature "allowed-ext")                    ; EXTEN

compound-type = (
  "type" => ("object" .feature "1.1"),
  ? required: [+text],
  ? properties: named<dataqualities>,
)

jsonschema = (
 ? (("type" => "number" / "string" / "boolean" / "integer" / "array")
    // compound-type
    // (type: text .feature "type-ext")                      ; EXTENSIO
   )
 ? enum: [+ allowed-types]
 ? const: allowed-types
 ? default: allowed-types
 ; number/integer constraints
 ? minimum: number
 ? maximum: number
 ? exclusiveMinimum: bool / number      ; jso draft 4/7
 ? exclusiveMaximum: bool / number      ; jso draft 4/7
 ? multipleOf: number                   ; ISSUE: Do we need this?
 ; text string constraints
 ? minLength: number
 ? maxLength: number
 ? pattern: text                ; regexp
 ? format: "date-time" / "date" / "time"
          / "uri" / "uri-reference" / "uuid"
          / (text .feature "format-ext")                      ; EXTENS
 ; array constraints
 ? minItems: number
 ? maxItems: number
 ? uniqueItems: bool
 ? items: { ;;; ultimately, this will be mostly recursive, but, for now
            ;;; let's find out what we actually need
     ? sdfRef: sdf-pointer          ; import limited to the subset that
     ? description: text            ; long text (no constraints)
     ? $comment: text               ; source code comments only, no sema
     ; commonqualities, ; -- ISSUE: should leave this out for non-comple
     ? ((type: "number" / "string" / "boolean" / "integer") ; no "array"
        // compound-type
        // (type: text .feature "itemtype-ext")                    ;
       )
     ; jso subset
     ? minimum: number
     ? maximum: number
```

```
    ? enum: [+ any]
    ? format: text
    ? minLength: number
    ? maxLength: number
    EXTENSION-POINT<"items-ext">
  }
)
```

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

This appendix describes the syntax of SDF defined in Appendix A, but using a version of the description techniques advertised on json-schema.org [I-D.handrews-json-schema-validation].

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.

(The json-schema.org descriptions need to be regenerated after the converter has been upgraded to handle the group choices introduced in the latest CDDL.)

Acknowledgements

This draft is based on sdf.md and sdf-schema.json in the old one-data-model language repository, as well as Ari Keranen's "alt-schema" from the Ericsson Research ipso-odm repository (which is now under subdirectory sdflint in the one-data model tools repository).

Contributors

Ari Keränen
Ericsson
FI-02420 Jorvas
Finland

Email: ari.keranen@ericsson.com

Wouter van der Beek
Cisco Systems
Eastpoint Business Park
Alfie Byrne Road
Dublin 3
Ireland

Email: wovander@cisco.com

Authors' Addresses

Michael Koster (editor)
SmartThings
665 Clyde Avenue
Mountain View, 94043
United States of America

Phone: +1-707-502-5136

Email: Michael.Koster@smartthings.com

Carsten Bormann (editor)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org