ASDF
Internet-Draft
Intended status: Standards Track
Expires: 24 July 2026

C. Bormann
Universit辰t Bremen TZI
J. Romann
Universit辰t Bremen

20 January 2026

Instance Information for SDF
draft-ietf-asdf-instance-information-01

Abstract

This document specifies instance-related messages to be used in conjunction with the Semantic Definition Format (SDF) for Data and Interactions of Things (draft-ietf-asdf-sdf). Split into four "archetypes", instance-related messages are always governed by SDF models, strictly separating instance and class information. _Context_ information plays a crucial role both for lifecycle management and actual device interaction.

// This revision applies a major restructuring, reduces redundancy
// and clarifies some of the concepts that are used throughout the
// document. It also expands upon three experimental application
// scenarios for instance-related messages.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-asdf-instance-information/.

Discussion of this document takes place on the "A Semantic Definition Format for Data and Interactions of Things" (ASDF) Working Group mailing list (mailto:asdf@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/asdf/. Subscribe at https://www.ietf.org/mailman/listinfo/asdf/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-asdf/instance-information.

Status of This Memo

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

This Internet-Draft will expire on 24 July 2026.

Copyright Notice

Table of Contents

1. Introduction

The Semantic Definition Format for Data and Interactions of Things (SDF, [I-D.ietf-asdf-sdf]) is a format for domain experts to use in the creation and maintenance of data and interaction models in the Internet of Things.

SDF is an Interaction Modeling format, enabling a modeler to describe the digital interactions that a class of Things (devices) offers, including the abstract data types of messages used in these interactions.

SDF is designed to be independent of specific ecosystems that specify conventions for performing these interactions, e.g., over Internet protocols or over ecosystem-specific protocol stacks.

SDF does not define representation formats for the _Instance Information_ that is exchanged in, or the subject of such, interactions; this is left to the specific ecosystems, which tend to have rather different ways to represent this information.

This document discusses Instance Information in different types and roles. It defines an _abstraction_ of this, as an eco-system independent way to reason about this information. This abstraction can be used at a _conceptual_ level, e.g., to define models that govern the instance information. However, where this is desired, it also can be used as the basis for a concrete _neutral representation_ (Format) that can actually be used for interchange to exchange information and parameters for interactions to be performed. In either case, the structure and semantics of this information are governed by SDF Models.

This document is truly work in progress. It freely copies examples from the [I-D.ietf-asdf-sdf-nonaffordance] document that evolves in parallel, with a goal of further synchronizing the development where that hasn't been fully achieved yet. After the discussion

stabilizes, we'll need to discuss how the information should be distributed into the different documents and/or how documents should be merged.

1.1. Conventions and Definitions

The definitions of [RFC6690], [RFC8288], and [I-D.ietf-asdf-sdf] apply.

Terminology may need to be imported from [LAYERS].

Representation: As defined in Section 3.2 of RFC 9110 [STD97], but understood to analogously apply to other interaction styles than Representational State Transfer [REST] as well.

Message: A Representation that is exchanged in, or is the subject of, an Interaction. Messages are "data in flight", not instance "data at rest" (the latter are called "Instance" and are modeled by the interaction model).

Depending on the specific message, an abstract data model for the message may be provided by the sdfData definitions (or of declarations that look like these, such as sdfProperty) of an SDF model.

Deriving an ecosystem specific representation of a message may be aided by _mapping files_ [I-D.bormann-asdf-sdf-mapping] that apply to the SDF model providing the abstract data model.

Instantiation: Instantiation is a process that takes a Model, some Context Information, and possibly information from a Device and creates an Instance.

Instance: Anything that can be interacted with based on the SDF model. E.g., the Thing itself (device), a Digital Twin, an Asset Management system... Instances are modeled as "data at rest", not "data in flight" (the latter are called "Message" and actually are/have a Representation). Instances that relate to a single Thing are bound together by some form of identity. Instances become useful if they are "situated", i.e., with a physical or digital "address" that they can be found at and made the subject of an interaction.

Instance-related Message: A message that describes the state or a state change of a specific instance. (TBC -- also: do we need this additional term?)

Message Archetype: In the context of instance-related messages: A

message with specific content and effect, covering a wider set of different use cases. In this document, we are observing a total of four instance-related message archetypes: Snapshot Messages, Construction Messages, Delta Messages, and Patch Messages.

Proofshot: A message that attempts to describe the state of an Instance at a particular moment (which may be part of the context). We are not saying that the Proofshot _is_ the instance because there may be different ways to make one from an Instance (or to consume one in updating the state of the Instance), and because the proofshot, being a message, is not situated.

Proofshots are snapshots, and they are "proofs" in the photographic sense, i.e., they may not be of perfect quality. Not all state that is characteristic of an Instance may be included in a Proofshot (e.g., information about an active action that is not embedded in an action resource). Proofshots may depend on additional context (such as the identity of the Instance and a Timestamp).

An interaction affordance to obtain a Proofshot may not be provided by every Instance. An Instance may provide separate Construction affordances instead of simply setting a Proofshot.

Construction: Construction messages enable the creation of a digital Instance, e.g., initialization/commissioning of a device or creation of its digital twins. They are like proofshots, in that they embody a state, however this state needs to be precise so the construction can actually happen.

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 [BCP14] (RFC2119) (RFC8174) when, and only when, they appear in all capitals, as shown here.

2. Instance Information and SDF

The instantiation of an SDF model does not directly express an instance, which is, for example, a physical device or a digital twin. Instead, the instantiation produces an instance-related _message_, which adheres to a uniform message format and is always controlled by the corresponding SDF model. Depending on the recipient and its purpose, a message can be interpreted as a report regarding the state of a Thing or the instruction to change it when consumed by the recipient.

Taking into account previous revisions of this document as well as [I-D.ietf-asdf-sdf-nonaffordance], we identified two main dimensions for covering the potential use cases for instance-related messages:

1. the intended effect of a message, which can either be a report or an update of a Thing's state, and

2. the actual content of the message, which may be freestanding (without a reference to a previous message or state) or relative (with such a reference).

Based on these considerations (as illustrated by the systematization in Table 1), we can identify the following four message archetypes:

1. _Snapshot_ messages that may contain contain both affordance-related and context information, including information about a Thing's identity,

2. _Construction_ messages that trigger a Thing's initial configuration process or its commissioning,

3. _Delta_ messages that indicate changes that have occurred since a reference state report, and

4. _Patch_ messages that update the Thing's state.

	Content	
	Freestanding	Relative
(Intended) Effect / State Exposure	Snapshot	Delta
State Change	Construction	Patch

Table 1: Systematization of instance-related messages along the dimensions "Content" and "(Intended) Effect".

The uniform message format can be used for all four message archetypes. Appendix B specifies the formal syntax of instance-related messages that all normative statements as well as the examples in this document will adhere to. This syntax can serve to describe both the abstract structure and the concrete shape of the messages that can be used as a neutral form in interchange.

In the following, we will first outline a number of general principles for instance-related messages, before detailing the specific archetypes we define in this document. The specification text itself will be accompanied by examples that have been inspired by [I-D.ietf-asdf-sdf-nonaffordance] and [I-D.ietf-asdf-digital-twin] that each correspond with one of the four archetypes.

2.1. Axioms for instance-related messages

Instance-related messages can be messages that relate to a property, action, or event (input or output data), or they can be "proofshots" (extracted state information, either in general or in a specific form such as a context snapshot etc.).

Instance-related messages are controlled by a _model_ (class-level information), which normally is the interaction model of the device. That interaction model may provide a model of the interaction during which the instance-related message is interchanged (at least conceptually), or it may be a "built-in" interaction (such as a proofshot, a context snapshot, ...) that is implicitly described by the entirety of the interaction model. This may need to be augmented/composed in some way, as device modeling may be separate from e.g. asset management system modeling or digital twin modeling. Instance-related messages use JSON pointers into the model in order to link the instance-related information to the model.

Instance-related messages are conceptual and will often be mapped into ecosystem-specific protocol messages (e.g., a bluetooth command). It is still useful to be able to represent them in a neutral ("red-star") format, which we build here as an adaption of the JSON-based format of the models themselves. An ecosystem might even decide to use the neutral format as its ecosystem-specific format (or as an alternative format).

Instance-related messages may be plain messages, or they may be deltas (from a previous state) and/or patches (leading from a previous or the current state to a next state). Several media types can be defined for deltas/patches; JSON merge-patch [RFC7396] is already in use in SDF (for sdfRef) and therefore is a likely candidate. (Assume that some of the models will be using Conflict-free replicated data types (CRDTs) (https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type) to facilitate patches.)

2.2. Context Information

Messages always have context, typically describing the "me" and the "you" of the interaction, the "now" and "here", allowing deictic statements such as "the temperature here" or "my current draw".

Messages may have to be complemented by this context for interpretation, i.e., the context needed may need to be reified in the message (compare the use of SenML "n"). Information that enables interactions via application-layer protocols (such as an IP address) can also be considered context information.

For this purpose, we are using the sdfContext keyword introduced by [I-D.ietf-asdf-sdf-nonaffordance]. Note that sdfContext _could_ also be modelled via sdfProperty.

TODO: explain how [RFC9039] could be used to obtain device names (using urn:dev:org in the example).

Note that one interesting piece of context information is the model itself, including the information block and the default namespace. This is of course not about the device or its twin (or even its asset management), because models and devices may want to associate freely. Also note that multiple models may apply to the same device (including but not only revisions of the same models).

3. Message Format

The data model of instance-related messages makes use of the structural features of SDF models (e.g., when it comes to metadata and namespace information), but is also different in crucial aspects.

3.1. Information Block

The information block contains the same qualities as an SDF model and, additionally, a mandatory messageId to uniquely identify the message. Furthermore, Delta messages can utilize the previousMessageId in order to link two messages and indicate the state change.

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
messageId	string	Unique identifier of this instance-related message
previousMessageId	string	Identifier used to connect this instance-related message to a previous one
timestamp	string	Indicates the point in time this instance-related message refers to
features	array of strings	List of extension features used
$comment	string	Source code comments only, no semantics

Table 2: Qualities of the Information Block

3.2. Namespaces Block

Similar to SDF models, instance-related messages contain a namespaces block with a namespace map and the defaultNamespace setting. In constrast to models, including a namespace quality is mandatory as at least one namespace reference is needed to be able to refer to the SDF model the instance-related message corresponds with.

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 3: Namespaces Block

3.3. Instance-of Block

Distinct from SDF models are two instance-specific blocks, the first of which is identified via the sdfInstanceOf keyword. Via the model keyword, this quality defines the entry point the sdfInstance quality from the next section is referring to. Furthermore, via the patchMethod field, a patch algorithm different from JSON Merge Patch can be specified.

Quality	Type	Description
model	string	Defines the entry point for sdfInstance by pointing to an sdfObject or an sdfThing. Has to be based on a namespace identifier from the namespaces map.
patchMethod	string	Allows for overriding the default patch method (JSON Merge Patch) by providing a registered value.
$comment	string	Source code comments only, no semantics

Table 4: Instance-of Block

3.4. Instance Block

In the instance block, state information for properties, actions, and events as well as context information can be included. Depending on the archetype, this information will either be used to report a Thing’s current state, to report state _changes_, or to update state via a patch or reconfiguration.

In addition to the messageId and previousMessageId from the info block, we are able to refer to

* the point in time when the information regarding the device state has been captured (via the timestamp quality) and

* the device identity (via the thingId qualitity in the sdfInstance block).

Since we are using the sdfInstance keyword as an entry point at the location pointed to via the model specfied in sdfInstanceOf, the instance-related message has to follow the structure of this part of the model (although, depending on the archetype, information that has not changed or will not be updated can be left out.)

The alternating structure of the SDF model (e. g., sdfObject/envSensor/sdfProperty/temperature) is repeated within the instance-related message, with the top-level sdfObject or sdfThing being replaced by sdfInstance at the entry point. Note that we also have to replicate a nested structure via sdfThing and/or sdfObject if present in the referenced SDF model.

Quality	Type	Description
thingId	string	(Optional) identifier of the instance (e.g., a UUID)
sdfThing	map	Values for the thing entries in the referenced SDF definition
sdfObject	map	Values for the object entries in the referenced SDF definition
sdfContext	map	Values for the context entries in the referenced SDF definition
sdfProperty	map	Values for the properties in the referenced SDF definition
sdfAction	map	Values for the actions in the referenced SDF definition
sdfEvent	map	Values for the events in the referenced SDF definition

Table 5: Instance Block

4. Message Archetypes

Based on the common message format defined in Section 3 and the systematization from Table 1, we can derive a set of four archetypes that serve different use cases and recipients.

TODO: Decide whether we want to add specific CDDL schemas for the four archetypes via extension points in the "base schema"

4.1. Snapshot Messages

This instance-related message contains information on a Thing’s state, both in terms of context information and the state of individual affordances. In the message, the previousMessageId field in the information block MUST NOT be present. Furthermore, when transmitting this message in its JSON format, the content type application/sdf-snapshot+json MUST be indicated if supported by the protocol used for transmission.

Snapshot messages MAY only contain values for a _subset_ of all possible affordances and context information exposed by a Thing. Security-related aspects, e.g. regarding authentication and authorization, MUST be taken into account when issueing a state report for a requesting party.

In practical use, we can at least differentiate two use cases for snapshot messages. The corresponding message variants are (colloquially) referred to as "Context Snapshots" and "Proofshots".

Context Snapshots _only_ contain context information related to a Thing (indicated via the sdfContext quality). Figure 1 gives an example for this kind of instance-related message.

```
{
  "info": {
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a42",
    "timestamp": "2025-07-01T12:00:00Z"
  },
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensors"
  },
  "defaultNamespace": "models",
  "sdfInstanceOf": {
    "model": "sensors:#/sdfObject/envSensor"
  },
  "sdfInstance": {
    "thingId": "envSensor:abc123",
    "sdfContext": {
      "installationInfo": {
        "floor": 3,
        "mountType": "ceiling",
        "indoorOutdoor": "indoor"
      }
    }
  }
}
```

Figure 1: Example of an SDF context snapshot.

Proofshot Messages are supersets of context snapshots that may also include state information associated with a Thing’s _interaction affordances_(properties, actions, and events).

```
// Note that while the format for describing the state of properties
// is clearly governed by the schema information from the
// corresponding sdfProperty definition, it is still unclear how to
// best model the state of sdfActions and sdfEvents.
```

Figure 2 shows a proofshot that captures the state of a sensor. Here, every property and context definition of the corresponding SDF model (see Figure 11) is mapped to a concrete value that satisfies the associated schema.

```
{
  "info": {
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a42",
    "timestamp": "2025-07-01T12:00:00Z"
  },
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfInstanceOf": {
    "model": "sensors:#/sdfObject/envSensor"
  },
  "sdfInstance": {
    "thingId": "envSensor:abc123",
    "sdfContext": {
      "installationInfo": {
        "mountType": "ceiling"
      }
    },
    "sdfProperty": {
      "temperature": 23.124
    }
  }
}
```

Figure 2: SDF proofshot example.

4.2. Construction Messages

Construction messages are structurally equivalent to snapshot messages but may only contain context information. Furthermore, the recipient of a construction message is supposed to initiate a configuration or comissioning process upon recption. Construction messages MUST be indicated by the media type application/sfd-construction+json if possible.

A construction message for a temperature sensor might assign an identity and/or complement it by temporary identity information (e.g., an IP address); its processing might also generate construction output (e.g., a public key or an IP address if those are generated on device) which can be described via instance-related messages such as snapshot messages.

The creation of construction messages is linked to the invocation of a constructor that starts the actual construction process. In practice, these constructors are going to be modeled as an sdfAction, although the way the sdfAction is going to be used exactly is not entirely clear yet.

```
// Note that it is not quite clear what a destructor would be for a
// physical instance -- apart from a scrap metal press, but according
// to RFC 8576 we would want to move a system to a re-usable initial
// state, which is pretty much a constructor.
```

Figure 3 shows a potential SDF construction message that initializes a device, setting its manufacturer and firmwareVersion as context information. The construction message also assigns a thingId, the unit of reported temperature values, and an initial ipAddress that can be used with the interaction affordances that may be present in the corresponding SDF model.

```
{
  "info": {
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a42"
  },
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfInstanceOf": {
    "model": "sensors:#/sdfObject/envSensor"
  },
  "sdfInstance": {
    "thingId": "envSensor:unit42",
    "sdfContext": {
      "ipAddress": "192.168.1.5",
      "unit": "Cel",
      "deviceIdentity": {
        "manufacturer": "HealthTech Inc.",
        "firmwareVersion": "1.4.3"
      }
    }
  }
}
```

Figure 3: Example for an SDF construction message

A special type of construction message that only contains identity-related information may be called an _Identity Manifest_. Figure 4 shows an example of an identity manifest that is structurally identical to the construction message from Figure 3, with the non-identity-related information left out.

Via sdfRequired, an SDF model can indicate which context information must be present and therefore initialized within an instance. All definitions included in sdfRequired MUST also be present in a construction message, while other sdfContext definitions could be left out.

```
{
  "info": {
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a42"
  },
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfInstanceOf": {
    "model": "sensors:#/sdfObject/envSensor"
  },
  "sdfInstance": {
    "thingId": "envSensor:unit42",
    "sdfContext": {
      "deviceIdentity": {
        "manufacturer": "HealthTech Inc.",
        "firmwareVersion": "1.4.3"
      }
    }
  }
}
```

Figure 4: Example of an SDF identity manifest

4.3. Delta Messages

Delta messages describe updates to a Thing’s state relative to a previous message. For this purpose, a previousMessageId MUST be present in the info block. When transmitting delta messages, the media type application/sdf-delta+json MUST be used if possible.

By default, the values contained in the message are applied to the preceding message(s) via the JSON Merge Patch algorithm. Via the patchMethod quality, different patch algorithms MAY be indicated.

Figure 5 shows an example Delta message that reports state changes compared to the ones reported in the previous message (identified via its previousMessageId). In this example, only the temperature that has been measured by the sensor has changed, which is why it is the only piece of information that is included.

Delta messages could be used in the Series Transfer Pattern [STP], which may be one way to model a telemetry stream from a device.

```
{
  "info": {
    "title": "Example SDF delta message",
    "previousMessageId": "026c1f58-7bb9-4927-81cf-1ca0c25a857b",
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a42"
  },
  "namespace": {
    "cap": "https://example.com/capability/cap",
    "models": "https://example.com/models"
  },
  "defaultNamespace": "cap",
  "sdfInstanceOf": {
    "model": "models:/sdfObject/envSensor"
  },
  "sdfInstance": {
    "sdfProperty": {
      "temperature": 24
    }
  }
}
```

Figure 5: Example of an SDF instance-related message that serves as a delta.

4.4. Patch Messages

Patch messages are structurally equivalent to delta messages, but once again are only allowed to contain context information. They utilize a patch _mechanism_ (which may be explicitly indicated via the patchMethod quality) to _alter_ the state of a Thing instead of _reporting_ state changes. Since patch messages are not referring to a preceding message, a previosMessageId MUST NOT be present in the information block. When transmitting state patches, the media type application/sdf-patch+json MUST be used if possible.

An example Patch Message is shown in Figure 6, where a change of the device’s mountType is signalled.

```
{
  "info": {
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a42"
  },
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfInstanceOf": {
    "model": "sensors:#/sdfObject/envSensor",
    "patchMethod": "merge-patch"
  },
  "sdfInstance": {
    "sdfContext": {
      "installationInfo": {
        "mountType": "wall"
      }
    }
  }
}
```

Figure 6: Example of an SDF context patch message that uses the common instance-related message format.

Practical uses for patch message include digital twins [I-D.ietf-asdf-digital-twin], where changes to physical attributes (such as the location) need to be reflected in the digital representation of a Thing.

5. Application Scenarios

The instance-related message format and the four architectures are usable in a number of use cases, some of which we are going to specify in the following. Other specifications may define additional use cases instance-related messages can be used for.

5.1. Construction

In SDF models, we can speicify a Thing’s configurable parameters via sdfContext definitions for which Construction Messages can provide concrete values. Figure 7 shows an example for such an SDF model. The parameters settable during construction (in this case: the temperature property’s unit) are modeled as sdfContext definitions, to which the entries of the sdfParameters map may point to using JSON pointers.

```
{
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfObject": {
    "sensor": {
      "sdfRequired": [
        "ipAddress",
        "deviceIdentity"
      ],
      "sdfContext": {
        "ipAddress": {
          "type": "string"
        },
        "unit": {
          "type": "string"
        },
        "deviceIdentity": {
          "type": "object",
          "properties": {
            "manufacturer": {
              "type": "string"
            },
            "firmwareVersion": {
              "type": "string"
            }
          }
        }
      },
      "sdfProperty": {
        "temperature": {
          "type": "number",
          "sdfParameters": {
            "unit": "#/sdfObject/sensor/sdfContext/unit"
          },
          "sdfRequired": "#/sdfObject/sensor/sdfContext/unit"
        }
      }
    }
  }
}
```

Figure 7: Example for SDF model with constructors

Based on the SDF model above, a Construction Message such as the one shown in Figure 3 can trigger a construction process. As indicated via sdfRequired, this process must include the initialization of an IP address as well as the device's identity definitions. In the example model, initializing the unit context definition is only required if the temperature property is present, which is expressed by the JSON pointer within the property's sdfRequired definition.

5.2. Protocol Binding Information

When using the sdfProtocolMap concept introduced in [I-D.ietf-asdf-sdf-protocol-mapping], some protocols may need context information such as a hostname or an IP address to actually be usable for interactions. This corresponds with the fact that the parameters related to application-layer protocols are often _class-level_ information and therefore not necessarily instance-specific.

For example, all instances of a smart light may use similar CoAP resources, with the only difference being the concrete IP address assigned to them. Therefore, we can utilize context information that varies between instances to complement the model information provided via an sdfProtocolMap.

Figure 8 illustrates the potential relationship between the two concepts in an SDF model. Here, a (hypothetical) CoAP protocol mapping specification defines an interface for parameters such as an IP address. Via JSON pointers, the sdfParameters within the sdfProtocolMap are linked to compatible sdfContext entries that may further restrict the set of allowed values via their schema definitions.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfObject": {
    "sensor": {
      "sdfContext": {
        "ipAddress": {
          "type": "string"
        }
      },
      "sdfProperty": {
        "temperature": {
          "type": "number",
          "sdfProtocolMap": {
            "coap": {
              "sdfParameters": {
                "ipAddress": "#/sdfObject/sensor/sdfContext/\
                                                        ipAddress"
              },
              "read": {
                "method": "GET",
                "href": "/temperature",
                "contentType": 60
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 8: Example of an SDF model where a CoAP-based protocol map points to the definition of relevant context information: an IP address.

Figure 9 shows how a Snapshot Message can provide the necessary IP address that is needed for retrieving the temperature value from the sensor described by the SDF model above.

```
{
  "info": {
    "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a47"
  },
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensor"
  },
  "defaultNamespace": "models",
  "sdfInstanceOf": {
    "model": "sensors:#/sdfObject/sensor"
  },
  "sdfInstance": {
    "sdfContext": {
      "ipAddress": "192.168.1.5"
    }
  }
}
```

Figure 9: Example of a snapshot message that provides the IP address needed to perform a CoAP-based interaction with the sensor from the previous figure.

5.3. Modelling the State of Interaction Affordances

Besides context information, Snapshot and (in a relative fashion) Delta Messages can report the current state associated with interaction affordances. For sdfProperty definitions, this is very similar to context information and very straightforward, as previously seen in in Figure 5.

Actions and events, however, are handled differently: In the case of actions, the state of one or more actions is reported, which might already be in a completed or error state, or may also still be running. For events, a history is reported that includes the returned values. The exact of number of action and event reports is implementation-dependent and may vary between deployments.

Figure 10 shows an example of a Snapshot Message for a lightswitch which reports the results of two toggle actions, one of which failed. The successfully completed action caused the emission of a toggleEvent with the same timestamp. As the lightswitch was turned on, the event was emitted with a value of true.

```
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   {
     "info": {
       "title": "Example SDF Snapshot Message with an Action and an \
                                                          Event History.",
       "messageId": "75532020-8f64-4daf-a241-fcb0b6dc4a85"
     },
     "namespace": {
       "cap": "https://example.com/capability/cap",
       "models": "https://example.com/models"
     },
     "defaultNamespace": "cap",
     "sdfInstanceOf": {
       "model": "models:/sdfObject/lightSwitch"
     },
     "sdfInstance": {
       "sdfAction": {
         "toggle": [
           {
             "timestamp": "2026-01-11T22:39:35.000Z",
             "status": "complete",
             "inputValue": null,
             "outputValue": null
           },
           {
             "timestamp": "2026-01-11T22:34:35.000Z",
             "status": "error",
             "inputValue": null,
             "outputValue": "Toggle failed.",
             "$comment": "This action completed with an error, which \
                                 is why an error message was returned."
           }
         ]
       },
       "sdfEvent": {
         "toggleEvent": [
           {
             "timestamp": "2026-01-11T22:39:35.000Z",
             "outputValue": true
           }
         ]
       }
     }
   }
```

Figure 10: Example of an SDF Snapshot Messages that reports an action and an event history.

6. Discussion

(TODO)

Discuss Proofshots of a Thing (device) and of other components.

Discuss concurrency problems with getting and setting Proofshots.

Discuss Timestamps appropriate for Things (Section 4.4 of [I-D.ietf-iotops-7228bis], [I-D.amsuess-t2trg-raytime]).

Discuss YANG config=true approach with regard to construction messages.

Discuss expressing a device’s "purpose of life" via context information

Discuss using context information to indicate provence

7. Security Considerations

* Pieces of instance-related information might only be available in certain scopes, e.g. certain security-related configuration parameters

(TODO)

8. IANA Considerations

TODO: Add media type registrations

9. References

9.1. Normative References

[BCP14] Best Current Practice 14, <https://www.rfc-editor.org/info/bcp14>. At the time of writing, this BCP comprises the following:

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>.

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>.

[I-D.ietf-asdf-sdf]
Koster, M., Bormann, C., and A. Ker辰nen, "Semantic Definition Format (SDF) for Data and Interactions of Things", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-25, 13 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-25>.

[I-D.ietf-asdf-sdf-nonaffordance]
Hong, J. and H. Lee, "Semantic Definition Format (SDF) Extension for Non-Affordance Information", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-nonaffordance-02, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-nonaffordance-02>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

[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>.

9.2. Informative References

[I-D.amsuess-t2trg-raytime]
Ams端ss, C., "Raytime: Validating token expiry on an unbounded local time interval", Work in Progress, Internet-Draft, draft-amsuess-t2trg-raytime-03, 19 October 2024, <https://datatracker.ietf.org/doc/html/draft-amsuess-t2trg-raytime-03>.

[I-D.bormann-asdf-sdf-mapping]
Bormann, C. and J. Romann, "Semantic Definition Format (SDF): Mapping files", Work in Progress, Internet-Draft, draft-bormann-asdf-sdf-mapping-07, 20 July 2025, <https://datatracker.ietf.org/doc/html/draft-bormann-asdf-sdf-mapping-07>.

[I-D.ietf-asdf-digital-twin]
Lee, H. and J. Hong, "Semantic Definition Format (SDF) modeling for Digital Twin", Work in Progress, Internet-

Draft, draft-ietf-asdf-digital-twin-03, 20 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-digital-twin-03>.

[I-D.ietf-asdf-sdf-protocol-mapping]
Mohan, R., Brinckman, B., and L. Corneo, "Protocol Mapping for SDF", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-protocol-mapping-03, 20 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-protocol-mapping-03>.

[I-D.ietf-iotops-7228bis]
Bormann, C., Ersue, M., Ker辰nen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-ietf-iotops-7228bis-03, 4 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-7228bis-03>.

[LAYERS] "Terminology for Layers", WISHI Wiki, <https://github.com/t2trg/wishi/wiki/NOTE:-Terminology-for-Layers>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/rfc6690>.

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

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021, <https://www.rfc-editor.org/rfc/rfc9039>.

[STP] Bormann, C. and K. Hartke, "The Series Transfer Pattern (STP)", Work in Progress, Internet-Draft, draft-bormann-t2trg-stp-03, 7 April 2020, <https://datatracker.ietf.org/doc/html/draft-bormann-t2trg-stp-03>.

Appendix A. Example SDF Model

Figure 11 shows the model all of the examples for instance-related messages are pointing to in this document. Note how the namespace is managed here to export the envSensor component into models:#/sdfObject/envSensor, which is the "entry point" used in the instance messages within the main document.

```
{
  "namespace": {
    "models": "https://example.com/models",
    "sensors": "https://example.com/sensors"
  },
  "defaultNamespace": "models",
  "sdfObject": {
    "envSensor": {
      "sdfContext": {
        "deviceIdentity": {
          "manufacturer": {
            "type": "string"
          },
          "firmwareVersion": {
            "type": "string"
          }
        },
        "installationInfo": {
          "type": "object",
          "properties": {
            "floor": {
              "type": "integer"
            },
            "mountType": {
              "enum": [
                "ceiling",
                "wall"
              ]
            }
          }
        }
      },
      "sdfProperty": {
        "temperature": {
          "type": "number",
          "unit": "Cel"
        }
      }
    }
  }
}
```

Figure 11: SDF Model that serves as a reference point for the instance-related messages in this draft

Appendix B. Formal Syntax of Instance-related Messages

```
   start = sdf-instance-message-syntax

   sdf-instance-message-syntax = {
    ; info will be required in most process policies
    ? info: sdfinfo
    namespace: named<text>
    ? defaultNamespace: text
    ? sdfInstanceOf: sdf-instance-of
    ? sdfInstance: sdf-instance
   }

   sdfinfo = {
    ? title: text
    ? description: text
    ? version: text
    ? copyright: text
    ? license: text
    ? messageId: text
    ; Identifier used to connect this instance message to a previous
    ; one:
    ; Allows this instance message to only contain values that have
    ; actually changed, turning it into a "Delta" or a "Patch",
    ; depending on the purpose of the message.
    ? previousMessageId: text
    ? timestamp: modified-date-time
    ? modified: modified-date-time
    ? features: [
                ]
    optional-comment
   }

   sdf-instance-of = {
    model: text
    ? patchMethod: text ; default is merge-patch
    optional-comment
   }

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

   ; 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 }

   commonqualities = (
    optional-comment
   )
```

```
   ; For describing the state of instances at a given point in time
   ;
   ; An sdfInstance can refer to either an sdfThing or an sdfObject.
   ; Structurally, it is mostly equivalent to that of an sdfThing
   ; with the additiona of a thingId quality.
   sdf-instance = (
       ? thingId: text

       thingqualities
   )
   objectqualities = {
    commonqualities

    cpaedataqualities
   }

   thingqualities = {
    sdfThing: named<thingqualities>

    sdfObject: named<objectqualities>

    commonqualities

    cpaedataqualities
   }

   cpaedataqualities = (
    ? sdfContext: named<allowed-types>

    ; Models the current state of the instance’s properties
    ? sdfProperty: named<allowed-types>

    ; Models the current state of the instance’s action affordances
    ;
    ; DISCUSS: How should the state of actions be modeled?
    ? sdfAction: named<any>

    ; Models an history for every event affordance
    ? sdfEvent: named<eventhistory>
   )

   eventhistory = [* eventqualities]

   eventqualities = {
       outputValue: allowed-types
       timestamp: modified-date-time
   }
```

```
   allowed-types = number / text / bool / null
                 / [* number] / [* text] / [* bool]
                 / {* text => any}

   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"]
   '
```

List of Figures

List of Tables

Acknowledgments

(TODO)

Authors' Addresses

Carsten Bormann
Universit辰t Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Jan Romann
Universit辰t Bremen
Germany
Email: jan.romann@uni-bremen.de