ASDF
Internet-Draft
Intended status: Standards Track
Expires: 22 January 2026

C. Bormann
Universitt Bremen TZI
J. Romann
Universitt Bremen

21 July 2025

Instance Information for SDF
draft-bormann-asdf-instance-information-05

Abstract

This document discusses types of Instance Information to be used in conjunction with the Semantic Definition Format (SDF) for Data and Interactions of Things (draft-ietf-asdf-sdf) and will ultimately define Representation Formats for them as well as ways to use SDF Models to describe them.

// The present revision 05 is intended as input for IETF 123, with a
// few observations from the Hackathon addressed and this status
// paragraph updated.

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-bormann-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 22 January 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 types of Instance Information and will ultimately define Abstract (eco-system independent) Representation Formats for them as well as ways to use SDF Models to describe them.

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.

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.

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]).

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.

Discuss YANG config=true approach.

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.

1.2. Terms we are trying not to use

Non-affordance: Originally a term for information that is the subject of interactions with other Instances than the Thing (called "offDevice" now), this term is now considered confusing as it would often just be an affordance of another Instance than the Thing.

2. Instance Information and SDF

Instantiation doesn’t produce an instance (ouch), which is the device, twin, etc., but a message.

2.1. Pre-structured types of messages

Pre-structured types of messages are those that relate to an SDF model in a way that, together with context and model, they are fully self-describing.

2.1.1. Input and output data of specific interactions

Messages always have context, typically describing the "me" and the "you" of the interaction, the "now" and "here", allowing deictic statements ("the temperature here", "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").

TODO: Use NIPC as an example how this could be used, including SCIM as a source of context information.

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

(Describe how protocol bindings can be used to convert these messages to/from concrete serializations...)

2.1.1.1. Examples for context information

```
{
  "namespace": {
    "models": "https://example.com/models",
    "boats": "https://example.com/boats"
  },
  "defaultNamespace": "boats",
  "sdfInstance": {
    "$context": {
      "$comment": "Potential contents for the SDF context",
      "deviceName": "urn:dev:org:30810-boat007",
      "deviceEui64Address": "50:32:5F:FF:FE:E7:67:28",
      "scimObjectId": "8988be82-50dc-4249-bed2-60c9c8797677",
      "parentInstance": "TODO -- addressing instance in data tree"
    }
  }
}
```

Figure 1: Example for an SDF instance with context information

2.1.2. Proofshots (read device, other component)

(See defn above.)

The following examples are based on Figure 2 of [I-D.lee-asdf-digital-twin-08], separated into an SDF proofshot and an SDF model.

A proofshot that captures the state of a boat with a heater is shown in Figure 2. Here, every property of the corresponding SDF model (see Figure 3) is mapped to a concrete value that corresponds with the associated schema information. The alternating structure of the SDF model (e. g., sdfThing/boot007/sdfObject/heater/sdfProperty/isHeating) is repeated in the proofshot, with sdfObject and sdfThing being replaced by sdfInstance.

While earlier approaches avoided the additional level of nesting by omitting the affordance quality names (i.e., sdfProperty, sdfAction, sdfEvent), including them explicitly avoids problems with namespace clashes and allows for a cleaner integration of meta data (via the $context keyword).

As in any instance message, information from the model is not repeated but referenced via a pointer into the model tree (sdfInstanceOf); the namespace needed for this is set up in the usual namespace section that we also have in model files.

Note that in this example, the proofshot also contains values for the implicit (offDevice) properties that are static (e.g., the physical location assigned to the instance) but are still part of the instance's proofshot as its location is fixed -- this boat apparently never leaves the harbor.

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

```
{
  "info": {
    "title": "A proofshot example for heater #1 on boat #007",
    "version": "2025-04-08",
    "copyright": "Copyright 2025. All rights reserved.",
    "proofshotId": "026c1f58-7bb9-4927-81cf-1ca0c25a857b"
  },
  "namespace": {
    "models": "https://example.com/models",
    "boats": "https://example.com/boats"
  },
  "defaultNamespace": "boats",
  "sdfInstance": {
    "boat007": {
      "sdfInstanceOf": "models:#/sdfThing/boat",
      "$comment": "Should the context be modeled via an additional \
      quality? Or should it rather become another kind of property?",
      "$context": {
        "scimObjectId": "a2e06d16-df2c-4618-aacd-490985a3f763"
      },
      "sdfProperty": {
        "identifier": "urn:boat:007:heater:1",
        "location": {
          "wgs84": {
            "latitude": 35.2988233791372,
            "longitude": 129.25478376484912,
            "altitude": 0
          },
          "postal": {
            "city": "Ulsan",
            "post-code": "44110",
            "country": "South Korea"
          },
          "w3w": {
            "what3words": "toggle.mopped.garages"
```

```
            }
          },
          "owner": "ExamTech Ltd."
        },
        "sdfInstance": {
          "heater": {
            "sdfInstanceOf": "models:#/sdfThing/boat/sdfObject/heater",
            "sdfProperty": {
              "characteristic": "12V electric heater, 800W, automatic \
                                                              cutoff",
              "status": "error",
              "report": "On February 24, 2025, the boat #007's heater \
                                  #1 was on from 9 a.m. to 6 p.m."
            },
            "sdfEvent": {
              "maintenanceSchedule": [
                {
                  "outputValue": "2025-04-10",
                  "timestamp": "2024-04-10T02:00:00Z"
                },
                {
                  "outputValue": "2026-04-10",
                  "timestamp": "2025-04-10T02:00:00Z"
                }
              ]
            }
          }
        }
      }
    }
```

Figure 2: SDF proofshot proposal for Figure 2 in [I-D.lee-asdf-digital-twin-08]

2.1.2.1. Corresponding SDF Model

Figure 3 shows a model like the one that could have been pointed to by the sdfInstanceOf pointers in the instance message. Note how the namespace is managed here to export the model components into models:#/sdfThing/boat and models:#/sdfThing/boat/sdfObject/heater.

(This example model only specifies structure; it also could come with semantic information such as the units that are used for wgs84 etc. In practice, the definition of wgs84 etc. probably would come from a common library and just be referenced via sdfRef.)

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

   {
     "info": {
       "title": "An example model of a heater on a boat",
       "version": "2025-01-27",
       "copyright": "Copyright 2025. All rights reserved."
     },
     "namespace": {
       "models": "https://example.com/models"
     },
     "defaultNamespace": "models",
     "sdfThing": {
       "boat": {
         "description": "A boat equipped with heating and navigation \
                                                                systems",
         "sdfProperty": {
           "identifier": {
             "$comment": "Is this actually off-device?",
             "type": "string",
             "offdevice": true
           },
           "owner": {
             "$comment": "Is this actually off-device?",
             "type": "string",
             "offdevice": true
           },
           "location": {
             "offdevice": true,
             "type": "object",
             "properties": {
               "wgs84": {
                 "type": "object",
                 "properties": {
                   "latitude": {
                     "type": "number"
                   },
                   "longitude": {
                     "type": "number"
                   },
                   "altitude": {
                     "type": "number"
                   }
                 }
               },
               "postal": {
                 "type": "object",
                 "properties": {
```

```
                "city": {
                  "type": "string"
                },
                "post-code": {
                  "type": "string"
                },
                "country": {
                  "type": "string"
                }
              }
            },
            "w3w": {
              "type": "object",
              "properties": {
                "what3words": {
                  "type": "string",
                  "format": "..."
                }
              }
            }
          }
        }
      },
      "sdfObject": {
        "heater": {
          "label": "Cabin Heater",
          "description": "Temperature control system for cabin \
                                                          heating",
          "sdfProperty": {
            "characteristic": {
              "description": "Technical summary of the heater",
              "type": "string",
              "default": "12V electric heater, 800W, automatic \
                                                            cutoff"
            },
            "status": {
              "description": "Current operational status",
              "type": "string",
              "enum": [
                "on",
                "off",
                "error"
              ],
              "default": "off"
            },
            "report": {
              "type": "string"
            }
```

```
            },
            "sdfEvent": {
              "maintenanceSchedule": {
                "$comment": "Should this actually be modeled as an \
                                                              event..?",
                "description": "Next scheduled maintenance date",
                "sdfOutputData": {
                  "type": "string",
                  "format": "date-time"
                }
              }
            }
          }
        }
      }
    }
  }
```

Figure 3: Revised SDF model proposal for Figure 2 of [I-D.lee-asdf-digital- twin-08]

2.1.3. Construction

Construction messages enable the creation of the 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.

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).

Construction messages need to refer to some kind of constructor in order to be able to start the actual construction process. It is still up for discussion whether this concept justifies a new keyword or whether construction and other lifecycle management processes should be modeled as sdfActions instead.

(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.)

2.1.3.1. Examples for SDF Constructors

This section contains examples for both approaches discussed above: Figure 4 introduces an sdfConstructor keyword which allows for defining both mandatory (in this example: temperatureUnit) and optional constructor parameters (in this example: ipAddress). The example shows that the names of constructor parameters may deviate from the quality names in the model (temperatureUnit vs unit) as the target quality is specified via a JSON pointer. Additionally, this constructor example explicitly labels the ipAddress as information that belongs to the $context of the proofshot.

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

{
  "info": {
    "title": "Example document for SDF (Semantic Definition Format) \
                                  with constructors for instantiation",
    "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": {
    "temperatureSensor": {
      "sdfProperty": {
        "temperature": {
          "description": "Temperature value measure by this Thing's \
                                                    temperature sensor.",
          "type": "number",
          "sdfParameter": {
            "unit": {
              "$comment": "Should schema information be settable \
via a constructor at all? This question might indicate that we need \
                                      different kinds of constructors",
              "type": "string"
            }
          }
        }
      },
      "sdfConstructor": {
        "construct": {
          "parameters": {
            "temperatureUnit": {
              "required": true,
```

```
                "target": "#/sdfObject/temperatureSensor/sdfProperty/\
                                                        temperature/unit"
              },
              "ipAddress": {
                "$comment": "Just trying some things out here. Should \
   this parameter target the context or rather an (offDevice?) property\
                                                                    ?",
                "required": false,
                "isContextInformation": true
              }
            }
          }
        }
      }
    }
  }
```

Figure 4: Example for SDF model with constructors

The alternative approach is shown in Figure 5. Here, the constructor is modeled as an sdfAction that contains the same set of parameters in its sdfInputData.

While this approach has advantages we do not need to introduce new keywords to achieve a similar functionality and can simply use a plain JSON object as the construction message a few things in this example are still unclear, especially when it comes to the mapping of constructor parameters to target affordances in the model and the designation of parameters as context information. Lastly, it is currently unclear what kind of schema information should be provided for the action's sdfOutputData. As a return value, a pointer to the instantiated device and/or the models describing it could make sense.

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

{
  "info": {
    "title": "Example document for SDF with actions as constructors \
                                                    for instantiation",
    "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": {
```

```
      "temperatureSensor": {
        "sdfProperty": {
          "temperature": {
            "description": "Temperature value measure by this Thing's \
                                                    temperature sensor.",
            "type": "number",
            "unit": {
              "$comment": "Should schema information be settable via \
  a constructor at all? This question might indicate that we need \
                                    different kinds of constructors",
              "type": "string"
            }
          }
        },
        "sdfAction": {
          "construct": {
            "sdfInputData": {
              "$comment": "DISCUSS: How can we establish a connection \
                between constructor parameters and target properties?",
              "type": "object",
              "properties": {
                "temperatureUnit": {
                  "type": "string",
                  "target": "#/sdfObject/temperatureSensor/sdfProperty\
                                                        /temperature/unit"
                },
                "ipAddress": {
                  "$comment": "How can we express that this is \
                                                 context information?",
                  "isContextInformation": true
                }
              },
              "required": [
                "temperatureUnit"
              ]
            },
            "sdfOutputData": {
              "type": "object",
              "properties": {
                "$comment": "DISCUSS: What kind of schema information \
                                               should we provide here?"
              }
            }
          }
        }
      }
    }
  }
```

Figure 5: Example for SDF model with constructors

2.1.3.2. Example for an SDF construction message

Figure 6 shows a potential SDF construction message that allows for the creation of a proofshot from a constructor that is contained within an SDF model.

Note that the ipAddress can be considered context information or an off-device property. TODO: Needs more discussion how to model this kind of information.

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

{
  "info": {
    "title": "Example SDF construction message",
    "$comment": "TODO: What kind of metadata do we need here?"
  },
  "namespace": {
    "cap": "https://example.com/capability/cap"
  },
  "defaultNamespace": "cap",
  "sdfConstruction": {
    "sdfConstructor": "cap:#/sdfObject/temperatureSensor/\
                                          sdfConstructors/construct",
    "arguments": {
      "temperatureUnit": "Cel",
      "ipAddress": "192.0.2.42"
    }
  }
}
```

Figure 6: Example for an SDF construction message

2.1.4. Deltas and Default/Base messages

What changed since the last proofshot?

What is different from the base status of the device?

Can I get the same (equivalent, not identical) coffee I just ordered but with 10 % more milk?

(Think merge-patch.)

A construction message may be a delta, or it may have parameters that algorithmically influence the elements of state that one would find in a proofshot.

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

{
  "info": {
    "title": "Example SDF delta construction message",
    "$comment": "TODO: What kind of metadata do we need here?"
  },
  "namespace": {
    "cap": "https://example.com/capability/cap"
  },
  "defaultNamespace": "cap",
  "sdfConstruction": {
    "sdfConstructor": "cap:#/sdfObject/temperatureSensor/\
                                          sdfConstructors/construct",
    "previousProofshot": "TODO (Can we provide an ID or just a \
                                                  timestamp here?)",
    "arguments": {
      "temperature": 24
    }
  }
}
```

Figure 7: Example for an SDF construction message for proofshot delta

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

A potential example for the use of proofshot deltas is shown in Figure 8. Here, the proofshot delta refers to the previous example in Figure 2 via its proofshotId, which is included in the previousProofshot quality. Compared to the previous proofshot, only the status property of the boat's heater has changed from error to operational. Via an algorithm such as JSON Merge Patch [RFC7396], the actual proofshot can be resolved by applying the delta to the previous version.

In future versions of this document, we will evaluate whether JSON Merge Patch is sufficient to fulfill the requirements for the resolution algorithm which have to formulated and refined themselves.

```
{
  "info": {
    "title": "Example SDF delta proofshot",
    "previousProofshot": "026c1f58-7bb9-4927-81cf-1ca0c25a857b",
    "proofshotId": "75532020-8f64-4daf-a241-fcb0b6dc4a42",
    "version": "2025-04-08",
    "copyright": "Copyright 2025. All rights reserved."
  },
  "namespace": {
    "models": "https://example.com/models",
    "boats": "https://example.com/boats"
  },
  "defaultNamespace": "boats",
  "sdfInstance": {
    "boat007": {
      "sdfInstanceOf": "models:#/sdfThing/boat",
      "sdfInstance": {
        "heater": {
          "sdfInstanceOf": "models:#/sdfThing/boat/sdfObject/heater",
          "sdfProperty": {
            "status": "operational"
          }
        }
      }
    }
  }
}
```

Figure 8: Example for an SDF proofshot delta that overrides the status property of the boat’s heater.

2.2. Metadata

One interesting piece of offDevice information is the model itself, including sdfinfo and the defaultnamespace. 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. Multiple models may apply to the same device (including but not only revisions of the models).

3. Discussion

(TODO)

4. Security Considerations

(TODO)

5. IANA Considerations

(TODO)

6. References

6.1. 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. Kernen, "Semantic Definition Format (SDF) for Data and Interactions of Things", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-23, 17 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-23>.

[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>.

6.2. Informative References

[I-D.amsuess-t2trg-raytime]
Amsss, 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-iotops-7228bis]
Bormann, C., Ersue, M., Kernen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-ietf-iotops-7228bis-02, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-7228bis-02>.

[I-D.lee-asdf-digital-twin-08]
Lee, H., Hong, J., Youn, J., and Y. Hong, "Semantic Definition Format (SDF) modeling for Digital Twin", Work in Progress, Internet-Draft, draft-lee-asdf-digital-twin-08, 14 June 2025, <https://datatracker.ietf.org/doc/html/draft-lee-asdf-digital-twin-08>.

[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. Roads Not Taken

This appendix documents previous modelling approaches that we eventually decided against pursuing further. Its main purpose is to illustrate our development process, showing which kind of alternatives we considered before choosing a particular way to describe instance information. We will remove this appendix as soon as this document is about to be finished.

A.1. Using SDF Models as Proofshots

As shown in Figure 9, the proofshot format could have also been modeled via SDF models where all sdfProperty definitions are given constvalues. However, this concept is not capable of capturing actions and events.

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

{
  "info": {
    "title": "An example model of the heater #1 in the boat #007 (\
                                          that resembles a proofshot)",
    "version": "2025-07-15",
    "copyright": "Copyright 2025. All rights reserved."
  },
  "namespace": {
    "models": "https://example.com/models"
  },
  "defaultNamespace": "models",
  "sdfThing": {
    "boat007": {
      "label": "Digital Twin of Boat #007",
      "description": "A ship equipped with heating and navigation \
                                                              systems",
      "sdfProperty": {
        "identifier": {
          "offDevice": true,
```

```
            "type": "string",
            "const": "urn:boat:007:heater:1"
          },
          "location": {
            "offDevice": true,
            "type": "object",
            "const": {
              "wgs84": {
                "latitude": 35.2988233791372,
                "longitude": 129.25478376484912,
                "altitude": 0.0
              },
              "postal": {
                "city": "Ulsan",
                "post-code": "44110",
                "country": "South Korea"
              },
              "w3w": {
                "what3words": "toggle.mopped.garages"
              }
            }
          },
          "owner": {
            "offDevice": true,
            "type": "string",
            "default": "ExamTech Ltd.",
            "const": "ExamTech Ltd."
          }
        },
        "sdfRequired": "#/sdfThing/boat007/sdfObject/heater1",
        "sdfObject": {
          "heater": {
            "label": "Cabin Heater",
            "description": "Temperature control system for cabin \
                                                          heating",
            "sdfProperty": {
              "characteristic": {
                "description": "Technical summary of the heater",
                "type": "string",
                "default": "12V electric heater, 800W, automatic \
                                                           cutoff",
                "const": "12V electric heater, 800W, automatic cutoff"
              },
              "status": {
                "description": "Current operational status",
                "type": "string",
                "enum": [
                  true,
```

```
                  false,
                  "error"
                ],
                "default": false,
                "const": false
              },
              "report": {
                "type": "string",
                "const": "On February 24, 2025, the boat #007's \
                                heater #1 was on from 9 a.m. to 6 p.m."
              }
            },
            "sdfEvent": {
              "overheating": {
                "$comment": "Note that it is unclear how to properly \
   model events or event history with the approach illustrated by this \
                                                              example.",
                "maintenanceSchedule": "Next scheduled maintenance \
                                                                  date",
                "sdfOutputData": {
                  "type": "string",
                  "format": "date-time",
                  "const": "2025-07-15T07:27:15+0000"
                }
              }
            }
          }
        }
      }
    }
```

Figure 9: SDF instance proposal for Figure 2 in [I-D.lee-asdf-digital-twin-08]

A.1.1. Alternative Instance Keys

Below you can see an alternative instance modelling approach with IDs as (part of the) instance keys.

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

```
{
  "info": {
    "title": "A proofshot example for heater #1 on boat #007",
    "version": "2025-07-15",
    "copyright": "Copyright 2025. All rights reserved.",
    "proofshotId": "026c1f58-7bb9-4927-81cf-1ca0c25a857b"
```

```
    },
    "namespace": {
      "models": "https://example.com/models",
      "boats": "https://example.com/boats"
    },
    "defaultNamespace": "boats",
    "sdfInstance": {
      "models:#/sdfThing/boat/007": {
        "sdfInstanceOf": "models:#/sdfThing/boat",
        "heater": "models:#/sdfThing/boat/sdfObject/heater/001",
        "$context": {
          "scimObjectId": "a2e06d16-df2c-4618-aacd-490985a3f763"
        },
        "identifier": "urn:boat:007:heater:1",
        "location": {
          "wgs84": {
            "latitude": 35.2988233791372,
            "longitude": 129.25478376484912,
            "altitude": 0
          },
          "postal": {
            "city": "Ulsan",
            "post-code": "44110",
            "country": "South Korea"
          },
          "w3w": {
            "what3words": "toggle.mopped.garages"
          }
        },
        "owner": "ExamTech Ltd."
      },
      "models:#/sdfThing/boat/sdfObject/heater/001": {
        "characteristic": "12V electric heater, 800W, automatic cutoff\
                                                                    ",
        "status": "error",
        "report": "On February 24, 2025, the boat #007's heater #1 \
                                    was on from 9 a.m. to 6 p.m.",
        "sdfEvent": {
          "maintenanceSchedule": [
            {
              "outputValue": "2025-04-10",
              "timestamp": "2024-04-10T02:00:00Z"
            },
            {
              "outputValue": "2026-04-10",
              "timestamp": "2025-04-10T02:00:00Z"
            }
          ]
```

```
        }
      }
    }
  }
```

Figure 10: SDF instance proposal (with IDs as part of the instance keys) for Figure 2 in [I-D.lee-asdf-digital-twin-08]

Acknowledgments

(TODO)

Authors' Addresses

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

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