ASDF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 14 September 2026

E. Ramos
Ericsson

13 March 2026

Semantic Definition Format (SDF) for API translation
draft-ramos-asdf-api-translation-02

Abstract

This document defines an extension to the Semantic Definition Format (SDF) that enables API translation between applications and devices. The translation enables clarification of complex syntax for a service or device to utilize a different API from the one that was first designed to use.

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 14 September 2026.

Table of Contents

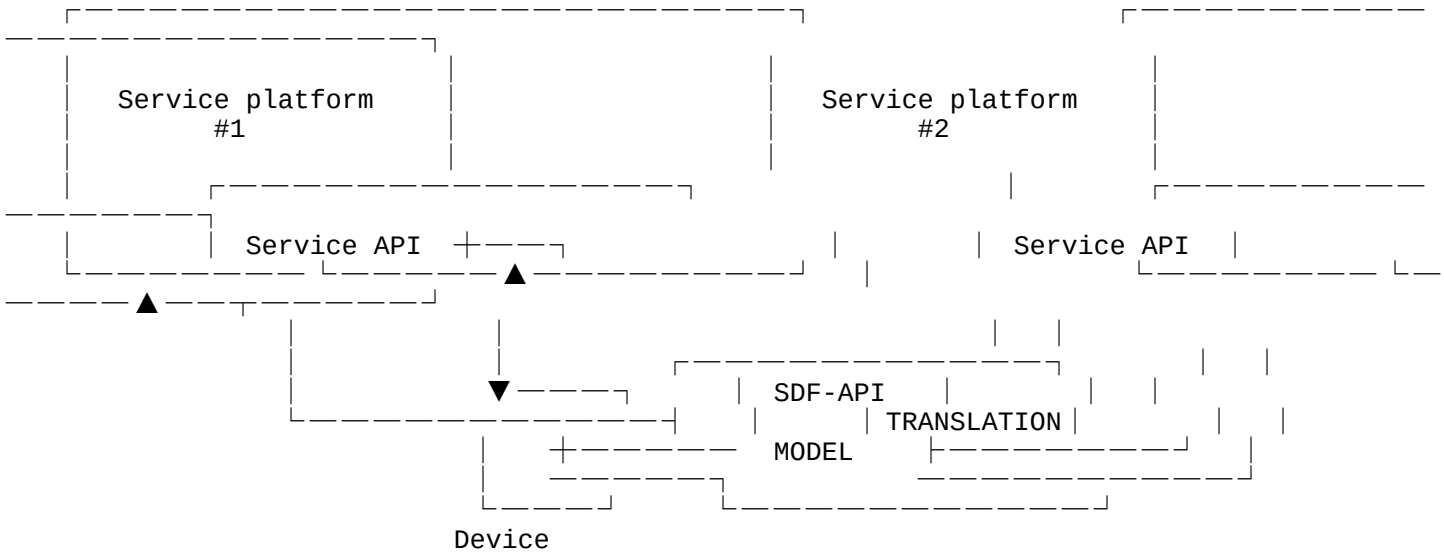
1. Introduction

In many cases, a service needed by an application can be provided through APIs by several suppliers. Semantic matching can be used to map the needs of the applications and adapt API calls to what the service suppliers could provide. The main purpose of this document is to describe methods that enable the translation of one device or application API to another equivalent API, utilizing syntax understandable by the deploying party.

An API translator from the original API to the updated one or even mapping to a new service would allow an automatic adaptation to the target API. Also, it would enable the possibility to adapt to multiple APIs with the same data model, enabling the deployment of services from multiple sources. This feature enables use cases such as the usage of the cheapest or most robust service, deployment of national services when traveling, and execution of device services even from a different ecosystem origin.

This document provides enabling elements for communication from devices or applications belonging to different ecosystems that do not share the same semantic modelling. Utilizing SDF syntax enhanced with an API format depiction, it is possible to provide an API description that is understandable with the syntax known by the device or application, if such semantics are defined in their data model. The final result uses a navigable description of the API's functions to provide the necessary information that is required by a device application to map external required services, their parameters, and results to an application logic using ontologically defined objects that can be matched to remote calls in another device application, for example.

The API requires a mapping in applications functions by matching syntax in addition to the actual discovery of the API; therefore, the importance of the translation to an understandable model. This matching could be enabled by several mechanisms described, and it is dynamically adjusted to every new interface faced. Using SDF in conjunction with the given schema allows an automated way to match functionality to application intents.



Example of a device using the API translation model to access the same service in a different platform with a different API

1.1. Terminology

The definitions of [I-D.ietf-asdf-sdf] apply. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. As explained in [RFC8174] the words apply when, and only when, they appear in all capitals, as shown here.

2. Discoverable API interfaces

A device or application may expose its capabilities as an API list. This could be acquired in real-time by calling a command that provides such a list of commands. For example, the resource “.well-known/commands” [RFC8615] could be used as a specific way to request all the commands available. The reply then includes a list of the different API supported calls, which would be called in a hierarchical way from the root domain that is provided by the device, an application, or a mediating platform. The reply would contain a description of the supported API with a semantic definition attached. Using the Semantic Definition Format (SDF), a reply for two commands may include what is given in the following example:

```
{
  "namespace": {
    "name": "Relevant_URL_or_namespace_used_as_semantic_reference_(example)",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns",
    "demo": "https://example.com/iss-demo-setup/#",
  },
  "defaultNamespace": "demo",
  "sdfObject": {
    "target_object": {
      "sdfProperty": {
        "target_property": {
          "description": "possible description of the property",
          "sdfRelation": {
            "FeedType": {
              "relType": "rdf:type",
              "target": [
                "demo:Output"
              ]
            }
          },
          "contentFormat": "application/typeofcontent-info"
        }
      },
      "sdfAction": {
        "Name_of_the_command_1": {
```

```
      "description": "Description of the command and input required",
      "sdfRelation": {
        "FeedType": {
          "relType": "rdf:type",
          "target": "demo:command1"
        }
      },
      "sdfInputData": {
        "type": "object",
        "properties": {
          "input_1_required": {
            "description": "Description of what element is needed as first input",
            "sdfRelation": {
              "FeedType": {
                "relType": "rdf:type",
                "target": [
                  "demo:input1"                    ]
              }
            },
            "type": "number",
            "sdfChoice": {
              "input_2_required": {
                "description": "A number needed as an input of the command",
                "const":"0",
                "sdfRelation": {
                  "FeedType": {
                    "relType": "rdf:type",
                    "target": [
                      "demo":"input2",
                    ]
                  }
                }
              }
            }
          }
        },
        "sdfOutputData": {
          "contentFormat": "application/typeofcontent-info"
        }
      },
      "Name_of_the_command_2": {
        "sdfRelation": {
          "FeedType": {
            "relType": "rdf:type",
            "target": "demo:command2"
          }
        },
```

```
            "sdfInputData": {
              "description": "Description of input required for second command",
              "sdfRelation": {
                "FeedType": {
                  "relType": "rdf:type",
                  "target": "demo:secondInput"
                }
              },
              "type": "typeoftheinput",
              "contentFormat": "application/typeofinput-info"
            }
          }
        }
      }
    }
  }
```

In this example, one service is defined with a command1, which requires an input1 (an object) and input2 (a number). The other service requires a different call (command2), which requires only one input (secondInput).

Further interaction capabilities can be included either in the initial response, as shown above in the "sdfAction" block, or could be discovered by further requests. For example, the descriptions of the two actions detailed above show if they require input, what the details of such input and the output are, and if there is additional information related to the nature of the output. Also, the description field could be tailor-made to provide a URI to a vocabulary that is understandable by the device or application. In this way, the device or application could make sense of the possible options, parameters, and actions that the command provides or requires.

3. Inputting parameters to the API

Once the available function calls and input parameters required by the API have been provisioned to the requester device or application, it can make calls to the API when required by its application logic. The application would map the “need for a capability” to the concrete capability provided by the provider application or device through matching the syntax and data model. The functions are then mapped to the calls that the application can handle and has defined syntax. The parts of the API that are not understood by the requester device are normally ignored. The mapping semantics from a target device or application may be communicated to the application of the requesting device using an SDF mapping extension draft [bormann-mapping]. In such mapping, the actions and data can be expressed using the data

model that are understandable by the requester and allows it to produce the API call and provide the required data. The only problem with this mapping is that if the API call structure is different, the logic required to call the target API might be complicated for certain device applications.

With SDF notation, it is possible to explain the data model but not the mapping for a specific API call. That would require, in some cases, mapping of values and, depending on the API definition, reshuffling of the parameters or even the URI utilized.

To tackle those challenges, we introduced a new SDF quality that can express URL template patterns from [RFC6570]. The quality *"template-href"*, includes the URL or the URL pattern that matches the command known by the requester device application. As explained in [RFC6570], the pattern may have variable terms, which are also given an SDF definition and relationship so the requested device can understand how to map its own data model to what the target API requires. Also, we introduce a value mapper *"template-value"* that enables mapping of values to parameters in the target API. The *sdfRelation* [I-D.ietf-asdf-sdf] is used as a vehicle to support the mapping between APIs.

The namespace *"cmd"* MUST be utilized to provide this API mapping values. In this example, we map a command_1 with several inputs to a command_2 that utilizes the inputs in a different way. The schema maps semantically the inputs and instructs the system how to make the API call based on the previously known API.

```
  {
 "info":{
    "title": "API mapping of two systems"
  },
  "namespace": {
    "targetSem": "https://example.org/targetSemantics/#",
    "targetDev": "https://example.org/targetDevice",
    "requestDev": "https://example.com/requesterDevice",
    "requestSem": "https://example.com/destinationSemantics/#"
  },
  "defaultNamespace": "requestSem",
  "sdfData": {
    "input_2_required": {
      "type": "number",
      "sdfChoice": {
        "requesterSemantic_1": {
          "const": 1
        },
        "requesterSemantic_2": {
```

```
            "const": 2
          }
        },
        "sdfRelation": {
          "FeedType": {
            "relType": "rdf:type",
            "target": [
              "targetDev:#/sdfObject/target_object/sdfAction/Name_of_the_command_1/sdfIn
putData/input2",
              "requestDev:#/sdfObject/request_object/sdfAction/requester_command/sdfInpu
tData/semanticRequesterInput"
            ]
          }
        }
      }
    },
    "map": {
      "#/sdfObject/request_object/sdfAction/requester_command": {
        "cmd:template-href": "https://example.org/targetDevice/control/Name_of_the_comma
nd_1?input1={input_2_required}&input2={input_1}",
        "sdfRelation": {
          "FeedType": {
            "relType": "rdf:type",
            "target": [
              "targetDev:#/sdfObject/target_object/sdfAction/Name_of_the_command_1",
              "requestSem:requester_command"
            ]
          }
        }
      },
      "cmd :template-values": {
        "input2": "#/sdfData/input_1_required",
        "input1": "#/sdfObject/target_object/sdfAction/name_of_command1/sdfInputData/inp
ut_1"
      },
      "#/sdfObject/request_object/sdfAction/requester_command_2": {
        "cmd:template-href": "https://example.org/targetDevice/control/Name_of_the_comma
nd_2?secondInput={requesterSecondInput}",
        "sdfRelation": {
          "FeedType": {
            "relType": "rdf:type",
            "target": "targetDev:#/sdfObject/target_object/sdfAction/Name_of_the_command
_2"
          }
        }
      },
      "cmd:template-values": {
        "secondInput": "#/sdfObject/request_object/sdfAction/requester_command_2/sdfInpu
tData/requesterSecondInput"
      }
    }
  }
```

4. Methods on the API calls

Sometimes, only calling an API is not enough. For example, in some cases, the API call requires additional actions such as submitting a payload. To handle this dimension on the API calls, the *method* command has been introduced. This command SHALL enable the requested device to understand how to handle the API call, since in the previous example, the API URL has all the information required in the call itself. However, in some cases, the action necessary with the API may need to be specified in a manner that the requested device can understand. The keyword *method* is meant to describe how MUST be handled the API URL and what to expect of it. In some other examples, the *method* could result in an OBSERVE or another type of new action not yet specified, for example, how to address an XR-type application.

An example could be the use of an LWM2M camera [lwm2m-cameraObject]. To use this model, a payload has to be submitted in the POST method. The following example describes such a translation from one camera API to the LWM2M model.

```
{
    "info": {
      "title": "LwM2M Camera Device to Requester dev. Application API mapping"
    },
    "namespace": {
      "sdm": "https://sdm.wide.ad.jp/sdmo/#/",
      "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns",
      "oma": "https://onedm.org/ecosystem/oma",
      "dev": "http://traffic-light202125.traffic-center.fi/10340/0/",
      "app": "http://owner.mycar.fi/"
    },
    "defaultNamespace": "dev",
    "map": {
      "#/sdfObject/video/sdfAction/playStream": {
        "cmd:template-href": "http://traffic-light202125.traffic-center.fi/10340/0/101
",
        "cmd:method":"POST",
        "description": "Starts the recording of the camera",
        "sdfInputData": {
            "type": "number",
            "const":1
         },
        "sdfRelation": {
            "FeedType": {
            "relType": "rdf:type",
             "target": [
                "dev:#/sdfObject/Camera/sdfAction/Local_Recording_Control",
                "oma:#/sdfObject/10340/101",
```

```
                    "sdm:playStream"              ]
                }
              }
          },
          "#/sdfObject/video/sdfAction/StopStream": {
            "cmd:template-href": "http://traffic-light202125.traffic-center.fi/10340/0/101
",
            "cmd:method":"POST",
            "description": "Stops the recording of the camera",
            "sdfInputData": {
                  "type": "number",
                  "const":0
            },
            "sdfRelation": {
              "FeedType": {
              "relType": "rdf:type",
              "target": [
                "dev:#/sdfObject/Camera/sdfAction/Local_Recording_Control",
                "oma:#/sdfObject/10340/101",
                "sdm:stopStream"
              ]
              }
            }
          },
          "#/sdfObject/Camera/sdfProperty/Input-video-stream":{
            "cmd:template-href": "http://traffic-light202125.traffic-center.fi/10340/0/9",
            "sdfRelation": {
              "FeedType": {
                "relType": "rdf:type",
                "target": [
                  "dev:#/sdfObject/Camera/sdfProperty/Recording_Location",
                  "sdm:VideoMedia"
                ]
              }
            }
          }
        }
      }
    }
```

5. Matching API to application logic

Each application has a logic that requires either information from the API call or the execution of a service. With this translation tool, the application logic may need to consider the different types of APIs that may be present. For example, if the API requires a payload or the info is directly appended to the call. Also, on the way the API produces the output, and how to handle it.

6. Ontologies describing APIs

Every application domain has an ontology or vocabulary that indicates actions and their parameters. These actions can be translated to APIs but they do not necessarily explain the API itself. Just the result of the API action.

Some ontologies describing APIs may need to be involved in some cases. Such as W3C Web of Things (WoT) Thing Description (TD)[wot], Hydra Core Vocabulary (for hypermedia Web APIs) [hydra] and others with less focus in APIs but relevant like OWL S / WSMO (Semantic Web Services) [owls], schema.org Actions [schema] and OGC SensorThings API [ogc] .

7. IANA Considerations

This document has the following actions for IANA.

Note to RFC Editor: Please replace all occurrences of "{XXXX}" with the RFC number of this specification and delete this paragraph.

7.1. Media Type

IANA is requested to add the following Media-Type to the "Media Types registry [IANA.media-types]. -->

Name	Template	Reference
sdf-mapping+json	application/ sdf-mapping+json	RFC XXXX, section 5.1

Table 1: A media type for SDF mapping files

8. Security Considerations

Some wider issues are discussed in [RFC8576].

9. Acknowledgements

Thanks to Ari Kernen for the support in realizing this idea.

10. References

10.1. Normative References

[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-25, 13 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-25>.

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

[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/rfc/rfc2119>.

[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/rfc/rfc8174>.

10.2. Informative References

[bormann-mapping]
Romann, C. B. and J., "Semantic Definition Format (SDF): Mapping files", n.d., <https://datatracker.ietf.org/doc/draft-bormann-asdf-sdf-mapping/>.

[hydra] "Hydra Core Vocabulary", n.d., <https://www.hydra-cg.com/spec/latest/core/>.

[lwm2m-cameraObject]
"Camera - OMAobjectId: 10340 V1.0", n.d., <https://raw.githubusercontent.com/OpenMobileAlliance/lwm2m-registry/prod/10340.xml>.

[ogc] "OGC SensorThings API Standard", n.d., <https://www.ogc.org/standards/sensorthings/>.

[owls] "OWL-S: Semantic Markup for Web Service", n.d., <https://www.w3.org/submissions/OWL-S/>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

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

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[schema] "Schemas from schema.org", n.d., <https://schema.org/Action>.

[wot] "Web of Things (WoT) Thing Description 1.1", n.d., <https://www.w3.org/TR/wot-thing-description11/>.

Author’s Address

Edgar Ramos
Ericsson
Hirsalantie 11
FI- 02420 Jorvas, Kirkkonummi
Finland
Email: edgar.ramos@ericsson.com