



DPC: 25/30473057 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 31 March 2025

Origin: International

Latest date for receipt of comments: 24 May 2025

Project No. 2023/00898

Responsible committee: AUE/16 Data Communication (Road Vehicles)

Interested committees:

Title: Draft BS ISO 17978-3 Road vehicles - Service-oriented vehicle diagnostics (SOVD). Part 3: Application programming interface (API).

Please notify admin.start@bsigroup.com if you are aware of any keywords that might assist in clarifying or identifying the standard or of the content of this standard:

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 24 May 2025

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

E-mail: admin.start@bsigroup.com

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/ Figure/Table/Note	Type of comment	Comment (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



DRAFT International Standard

ISO/DIS 17978-3

Road vehicles — Service-oriented vehicle diagnostics (SOVD) —

Part 3: Application programming interface (API)

*Véhicules routiers – Diagnostic Véhicule Orienté Services
(SOVD) —*

Partie 3: Interface de Programmation D'applications (API)

ICS: 43.040.15

ISO/TC 22/SC 31

Secretariat: **DIN**

Voting begins on:
2025-03-31

Voting terminates on:
2025-06-23

This document is circulated as received from the committee secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/DIS 17978-3:2025(en)

© ISO 2025



Documents

Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	vii
Introduction	viii
1 Scope	1
2 Normative references	1
3 Terms and definitions	2
4 Symbols and abbreviated terms	3
4.1 Symbols	3
4.2 Abbreviated terms	3
5 SOVD API general aspects	4
5.1 Use cases overview	4
5.2 Accessing the API	5
5.2.1 HTTP REST based approach	5
5.2.2 Server-Sent Events	6
5.3 Entities	9
5.3.1 General	9
5.3.2 Entity types and collections	9
5.3.3 Entity hierarchy and relations	10
5.3.4 Entity identifier and path	11
5.3.5 Location of the SOVD server	12
5.3.6 Multi-SOVD server environments	12
5.4 Resources	13
5.4.1 General	13
5.4.2 Standardized resource collections	13
5.4.3 Standardized resources	14
5.4.4 Restricting access to entities and resources by authentication	15
5.4.5 Custom specific resource collections and types	15
5.5 Illustration of SOVD concepts for Entities and resources	16
5.6 SOVD API versioning	17
5.7 Internationalization and localization	18
5.7.1 Internationalization	18
5.7.2 Unit definitions	18
5.7.3 Internationalization of enumerations	19
5.7.4 Other localization topics (numbers and dates)	20
5.8 Response status codes	20
5.8.1 General	20
5.8.2 Generic definition of status codes	20
5.8.3 Definition of error details	21
5.8.4 SOVD error codes for common cases	22
5.9 Common primitive data types	23
5.10 Definition of JSON schema and objects for vehicle responses	23
5.11 Discovery of the SOVD server for the co-located use case cluster	26
5.11.1 General concept	26
5.11.2 Network connection	26
5.11.3 Address an SOVD server using mDNS	26
5.11.4 Discovery of the SOVD server using DNS-SD	26
5.11.5 Example	27
5.12 Discovery of the SOVD server for the remote use case cluster	27
6 Capability description	27
6.1 General	27
6.2 Capability description structure	27
6.2.1 General	27
6.2.2 Info object	28
6.2.3 Path item object	28

6.2.4	Operation object	29
6.2.5	Schema object	30
6.2.6	Header object	31
6.2.7	Query Parameter object	31
6.2.8	x-sovd-tags object	32
6.3	Online and offline capability descriptions	32
6.3.1	General	32
6.3.2	Offline capability description	33
6.3.3	Online capability description	33
6.4	Variant handling in offline capability description	33
6.4.1	General	33
6.4.2	Extension: x-sovd-applicability	33
6.4.3	Usage of x-sovd-applicability by the SOVD client	34
7	SOVD REST API	35
7.1	General	35
7.2	Definition of conventions for API methods	36
7.2.1	Query parameters	36
7.2.2	Request body	36
7.3	Timeout definitions	36
7.4	API methods for access of supported SOVD server versions	37
7.4.1	Read all supported versions of the SOVD server	37
7.5	API methods for access to capability description content	38
7.5.1	General	38
7.5.2	Retrieve online capability description	38
7.6	API methods for discovery of Entities and resources	39
7.6.1	General	39
7.6.2	Query for Entities from an SOVD server	39
7.6.3	Query for capabilities of an Entity	45
7.7	Enhancement of API methods for tagged Entities and resources	48
7.7.1	General	48
7.7.2	Query for tagged Entities and resources	48
7.7.3	Retrieve tags from Entity or a resource	49
7.7.4	Query for tagged capabilities of an Entity	51
7.8	API methods for fault handling	52
7.8.1	General	52
7.8.2	Query for faults from an Entity	52
7.8.3	Read a fault resource	56
7.8.4	Delete all faults of an Entity	58
7.8.5	Delete single fault of an Entity	58
7.9	API methods for data resource read / write access	59
7.9.1	General	59
7.9.2	Retrieve of data resource categories and groups of an Entity	60
7.9.3	Query for data from an Entity	63
7.9.4	Read a data resource	65
7.9.5	Read multiple data values from an Entity	67
7.9.6	Write a data resource	73
7.10	API methods for cyclic-subscriptions	74
7.10.1	General	74
7.10.2	Query for cyclic-subscriptions from an Entity	74
7.10.3	Issue a new cyclic-subscription	75
7.10.4	Retrieve details of a cyclic-subscription	78
7.10.5	Update a cyclic-subscription	79
7.10.6	Delete a cyclic-subscription	80
7.11	API methods for triggers	81
7.11.1	General	81
7.11.2	Issue a new trigger	83
7.11.3	Query for triggers from an Entity	89
7.11.4	Retrieve details of a trigger	90

7.11.5	Update a trigger	92
7.11.6	Delete a trigger	93
7.11.7	Trigger value reception	94
7.12	API methods for configuration	95
7.12.1	General	95
7.12.2	Query for configurations from an Entity	95
7.12.3	Read a configuration resource	97
7.12.4	Write a configuration resource	100
7.12.5	Reset configuration	103
7.13	API methods for clearing data	105
7.13.1	General	105
7.13.2	Retrieve list of supported clear-data types	106
7.13.3	Start clearing of data	107
7.13.4	Retrieve status of clearing data	108
7.14	API methods for control of operations	110
7.14.1	General	110
7.14.2	Ensuring co-Located diagnostics	110
7.14.3	Query for operations from an Entity	111
7.14.4	Query for executions of an operation	112
7.14.5	Retrieve details of an operation	113
7.14.6	Start execution of an operation	116
7.14.7	Read the status of an operation execution	118
7.14.8	Terminate the execution of an operation	120
7.14.9	Support for execute / freeze / reset and custom capabilities	121
7.15	API methods for script execution	123
7.15.1	General	123
7.15.2	Upload a script	123
7.15.3	Query for scripts from an Entity	125
7.15.4	Retrieve details of a script	126
7.15.5	Remove a script	128
7.15.6	Start execution of a script	129
7.15.7	Query for executions of a Script	132
7.15.8	Read the status of a script execution	133
7.15.9	Terminate the script execution	134
7.15.10	Remove the execution of a script	135
7.16	API methods for support of target modes	136
7.16.1	General	136
7.16.2	Query for modes of an Entity	137
7.16.3	Retrieve details of a mode	138
7.16.4	Explicit control of Entity states via their defined modes	140
7.16.5	Hint-based control of Entity states via their defined modes	141
7.17	API methods for locking	141
7.17.1	General	141
7.17.2	Acquire a lock on an Entity	144
7.17.3	Query for locks of an Entity	146
7.17.4	Retrieve details of a lock	148
7.17.5	Modify the expiration time of a lock on an Entity	149
7.17.6	Release an acquired lock on an Entity	150
7.18	API methods for software update	151
7.18.1	General	151
7.18.2	Query for updates from an Entity	153
7.18.3	Retrieve details of an update	155
7.18.4	Automated installation of an update	157
7.18.5	Prepare installation of an update	158
7.18.6	Execute installation of an update	159
7.18.7	Read the status of an update	161
7.18.8	Delete update package from an SOVD server	163
7.18.9	Register an update at the SOVD server	164
7.19	API methods for restarting an Entity	165

7.19.1	General	165
7.19.2	Read the status of an Entity	166
7.19.3	Start an Entity	167
7.19.4	Restart an Entity	169
7.19.5	Forced Restart an Entity	170
7.19.6	Shutdown an Entity	171
7.19.7	Forced Shutdown an Entity	172
7.20	API methods for handling of bulk-data	173
7.20.1	General	173
7.20.2	Retrieve bulk-data categories of an Entity	175
7.20.3	Retrieve details of bulk-data resource	176
7.20.4	Download and upload bulk-data	178
7.20.5	Delete bulk-data	182
7.21	API methods for logging	184
7.21.1	General	184
7.21.2	Retrieve resources of an SOVD log	184
7.21.3	Read log entries from an SOVD log	185
7.21.4	Configure SOVD log	188
7.21.5	Retrieve the current SOVD log configuration from an Entity	190
7.21.6	Reset SOVD logging configuration to default	191
7.22	API methods for communication-logs	192
7.22.1	General	192
7.22.2	Query for communication-logs from an Entity	192
7.22.3	Configure communication-logs for an Entity	193
7.22.4	Read the status of a communication-logs	195
7.22.5	Control a communication-log	196
7.22.6	Delete communication-logs	197
7.23	Authentication of SOVD clients (informative)	198
7.23.1	Online and offline authentication and authorization	198
7.23.2	Secure Connection using TLS	204
7.23.3	Verifying SOVD client credentials and requesting a token at the backend	204
7.23.4	Verifying SOVD client credentials at the ExVe	204
7.23.5	Requesting a token	205
7.23.6	Request header for access-restricted resources	206
7.23.7	Validating a token	206
8	Classic diagnostic adapter	207
8.1	General	207
8.2	Access to UDS based entities	207
8.3	Specific mapping of UDS services to SOVD modes	208
8.3.1	Mapping of DiagnosticSessionControl (10 ₁₆) subfunctions to SOVD modes	208
8.3.2	Mapping of SecurityAccess (27 ₁₆) subfunctions to SOVD modes	208
8.3.3	Mapping of Authentication (29 ₁₆) to SOVD modes	209
8.3.4	Mapping of CommunicationControl (28 ₁₆) subfunctions to SOVD modes	209
8.3.5	Mapping of ControlDTC Setting (85 ₁₆) subfunctions to SOVD modes	209
8.4	Mapping of UDS services to data resources	209
8.5	Mapping of UDS services to fault resources	209
8.6	Mapping of UDS services to operation resources	210
8.6.1	Mapping of Routine (31 ₁₆) and InputOutputControlByIdentifier (2F ₁₆) services to operation resources	210
8.6.2	DEPRECATED — Mapping of EcuReset service (11 ₁₆) to operation resources	210
8.7	Mapping of EcuReset service (11 ₁₆) to restart an Entity resource	210
Annex A (normative) Access to legally required diagnostics		211
Annex B (informative) SOVD API common example		214
Bibliography		224

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, Road Vehicles, Subcommittee SC 31, Data Communication.

A list of all parts in the ISO 17978 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The introduction of HPCs in vehicles is associated with changes in the classical electronic and electrical vehicle architectures. Besides classical distributed embedded ECU architectures, domain- or zone-based architectures are also available. These architectures also extend beyond the physical vehicle.

This extends the focus from checking hardware to also checking the functionality of applications. It requires the recording of data such as memory usage, processor load, and the number of active services, as well as the collection of log and trace files.

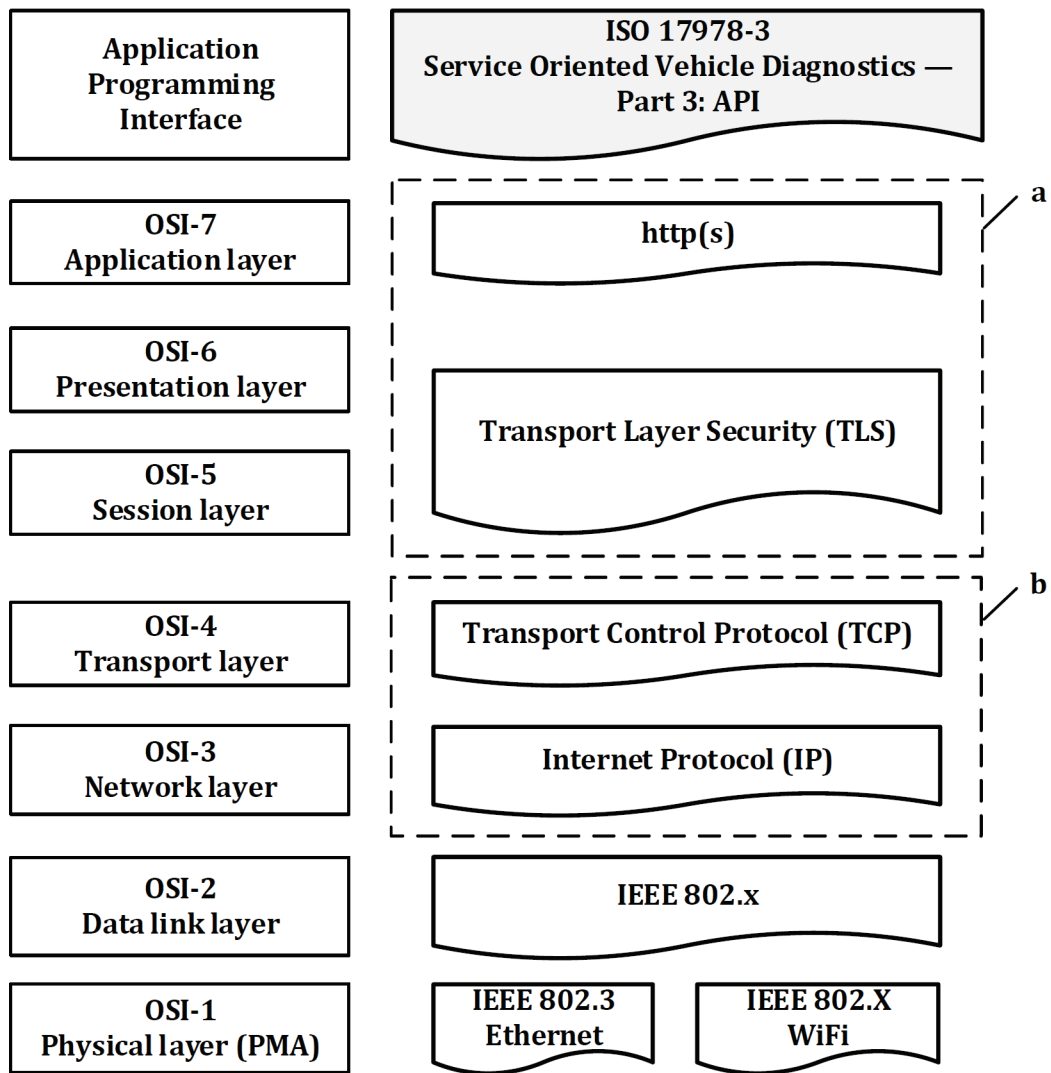
These topics result in challenges regarding the management of the vehicle life cycle. Some aspects to be considered are:

- Faster release and update cycles
- Increased requirements such as data protection and cybersecurity
- State-of-the-Art diagnostic API using current information technologies

The ISO 17978 series defines an API which standardizes the methods for:

- discovering the SOVD capabilities,
- performing diagnostics,
- (re-)configuring and re-programming,
- allowing the introduction of new functionalities

[Figure 1](#) shows the OSI layers of the ISO17978 series.

**Key**

- a communication protocol
 b network technology depending on E/E vehicle network architecture

Figure 1 — OSI layers of SOVD

For the convenience of the user, a machine-readable OpenAPI definition of the methods is published alongside this document in the ISO Standards Maintenance Portal: <https://standards.iso.org/iso/17978/>

NOTE This document marks some API methods as deprecated. Deprecated methods might be removed in a future version of this document. The background to this is the release of ASAM SOVD version 1.0 in 2022.

Road vehicles — Service-oriented vehicle diagnostics (SOVD) —

Part 3: Application programming interface (API)

1 Scope

This document defines an API which standardizes the methods for diagnosing HPCs and legacy ECUs, the retrieval of the diagnostic capabilities, and the discovery of the SOVD methods in an extended vehicle. The SOVD API provides a unified access to classic ECUs and HPCs.

The SOVD API provides functions such as:

- Access to the faults, including reading the fault entries, reading environment data, and deleting fault entries.
- Measurements and identifications of all entities.
- Execution of routines, I/O controls, and software functions in certain modes or states.
- Configuration of a vehicle (e.g., equipment, country, customer demand, variant coding etc.).
- Encapsulation of the ExVe manufacturer specific software update strategy (incl. FOTA if available).
- Access to logging information of an HPC.

With these features SOVD can cover all areas of the vehicle life cycle which includes in particular Engineering (Development), Manufacturing (Production), After Sales (Maintenance and Repair), Legal and Technical inspections, and Vehicle operation (Use).

There are several aspects which are not covered by this document, as they are specific to the implementation of an SOVD server, for example:

- Prevention and quick reaction to attack vectors like denial of service, zero-day exploits of vulnerabilities, ...
- Recognition of security incidents
- Maintaining the operational safety based on data monitoring
- Management of security incidents
- Data privacy issues
- Load balancing of concurrent requests

This document considers the extended vehicle as defined in ISO20077 series. This means that the API or its elements can be inside or outside the physical vehicle.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 17978-1, *Road vehicles — Service-oriented vehicle diagnostics (SOVD) — Part 1: SOVD General principles*

ISO 17978-2, *Road vehicles — Service-oriented vehicle diagnostics (SOVD) — Part 2: Use case definition*

HTML, *Server-sent-events*

IETF RFC 9110, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*

IETF RFC 1847, *Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted*

IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*

IETF RFC 3339, *Date and Time on the Internet: Timestamps*

IETF RFC 6782, *Multicast DNS*

IETF RFC 6783, *DNS-Based Service Discovery*

IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*

IETF RFC 6901, *JavaScript Object Notation (JSON) Pointer*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

IETF RFC 6570, *URI Template*

Open API Initiative, *OpenAPI Specification v3.1.0*

OpenJS Foundation, *JSON Schema 2020-12 Release Notes*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.1

Application

App

Entity (3.5) representing a software program that performs a specific task.

3.2

Area

Entity (3.5) representing a logical view on vehicle topologies.

3.3

ContributingEntity

Abstract *Entity* (3.5) that describes relationships like *hosts*, *is-located-on*, *contains*, *depends-on*, *belongs-to* to represent dependencies between specialized entities *Component* (3.4), *App* (3.6), and *Function* (3.6).

3.4

Component

Entity (3.5) representing hardware (characterized by the availability of a CPU) or software (characterized by the availability of an operating system or hypervisor), or a combination of both.

3.5**Entity**

object combining logically interrelated diagnostic information and represented in the form of resources.

Note 1 to entry: Entity is an abstract base class. The specific entities describe a diagnostic view of the vehicle.

Note 2 to entry: The term “Entity” used in the context of this document refers to “SOVD entity” (see ISO 17978-2).

3.6**Function**

Entity (3.5) representing a functional view on vehicle information spread across *Components* (3.4).

3.7**SOVDServer**

Entity (3.5) representing the server side implementation of the REST API services.

4 Symbols and abbreviated terms**4.1 Symbols**

— Empty table cell

4.2 Abbreviated terms

App	Application
C	Conditional
CDA	Classic Diagnostic Adapter
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DLT	Diagnostic Log and Trace
DNS	Domain Name System
DNS-SD	DNS Service Discovery
E/E	electric/electronic
ePTI	electronic Periodic Technical Inspection
FOTA	Firmware over the Air
ExVe	Extended Vehicle
HPC	High Performance Computer
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPs	HTTP Secure
IETF	Internet Engineering Task Force
IP	Internet Protocol

JSON	Java Script Object Notation
M	Mandatory
MD5	Message-Digest Algorithm 5
mDNS	Multicast Domain Name System
MIME	Multipurpose Internet Mail Extensions
O	Optional
OBD	Onboard Diagnostics
REST	Representational State Transfer
SHA256	Secure Hash Algorithm 256
SSE	Server-Sent Events
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TXT	TeXT strings
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VIN	Vehicle Identification Number
XML	eXtensible Markup Language
YAML	Yet Another Markup Language

5 SOVD API general aspects

5.1 Use cases overview

ISO 17978-2 defines use cases. For information, [Table 1](#) shows a mapping of those use cases to the API method calls as defined in this standard.

Table 1 — Mapping of ISO 17978-2 use cases to ISO 17978-3 API method calls

ISO 17978-2 use case #	ISO 17978-2 use case title	ISO 17978-3 API subclause(s) implementing this use case
Use Case 01	Discover SOVD Server(s)	5.11
Use Case 02	Discover Entities	7.6
Use Case 03	Authenticate SOVD Clients	7.23
Use Case 04	Read Fault(s) and/or its Details from an Entity	7.8.1 , 7.8.2 , 7.8.3
Use Case 05	Delete Fault(s) of an Entity	7.8.4 , 7.8.5
Use Case 06	Read Data Resources	7.9.1 , 7.9.2 , 7.9.3 , 7.9.4 , 7.9.5
Use Case 07	Write Data Items	7.9.6

Table 1 (continued)

ISO 17978-2 use case #	ISO 17978-2 use case title	ISO 17978-3 API subclause(s) implementing this use case
Use Case 08	Control Operations	7.14
Use Case 09	Read and Write Configurations	7.12
Use Case 10	Control Software Updates	7.18
Use Case 11	Handle Bulk Data	7.20
Use Case 12	Control Logging and Retrieve Logs	7.21
Use Case 13	Control Communication Logging	7.22
Use Case 14	Manage and Execute Scripts	7.15
Use Case 15	Manage Triggers and subscriptions	7.10 , 7.11
Use Case 16	Query an Online Capability Description	6.3.3 , 7.5
Use Case 17	Provide an Offline Capability Description	6.3.2

5.2 Accessing the API

5.2.1 HTTP REST based approach

The SOVD API uses the following standards and technologies:

- The API follows REST principles[\[13\]](#) and uses JSON[\[23\]](#) for encoding the transmitted data.
- SOVD is designed to work with HTTP/1.1[\[2\]](#) but for achieving the best communication performance HTTP/2[\[18\]](#) is recommended. No HTTP/2 specific features are used.
- The SOVD API utilizes the OpenAPI[\[7\]](#) specification to define the API as well as the diagnostic capabilities of the vehicle.
- It is recommended to build authentication and authorization upon OpenID Connect and OAuth 2.0,[\[9\]](#),[\[10\]](#),[\[11\]](#) but an ExVe manufacturer may use other authentication mechanism like certificates if required.

Based on REST principles, the diagnostic content is provided in the form of resources.

A resource is accessible via its specific resource path. In SOVD, the resource path is composed of the path to the individual **Entity** (see [5.3](#)) and the standardized resources and resource collections provided for that **Entity** (see [5.4](#)).

The operations available for the individual diagnostic content are represented using HTTP methods. [Table 2](#) shows the HTTP methods used by SOVD.

Table 2 — HTTP methods of the SOVD API

HTTP method	Purpose
GET	Reading content from a resource
PUT	Update content of a resource (e.g., by writing a new value)
POST	Creation of new (temporary) resources
DELETE	Deletion of created resources, resetting content to default
OPTIONS	The server returns which HTTP methods are supported for a particular resource.

An SOVD server shall support the method OPTIONS on each resource to inform a client about the supported methods. By this a client can determine which functionality (e.g., writing a data value) is supported for a specific resource. For example, for the following data resource representing the software identification of an ECU, the SOVD server returns that only the **GET** method is supported, besides **OPTIONS**.

Request:

```
OPTIONS {base_uri}/apps/WindowControl/data/swVersion HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
Allow: OPTIONS, GET
```

The example in [Figure 2](#) illustrates the REST based approach used by SOVD.

The example shows a software **Application** in the vehicle providing functionality to access information about the windows, for example it offers information about the “RearWindows” positions.

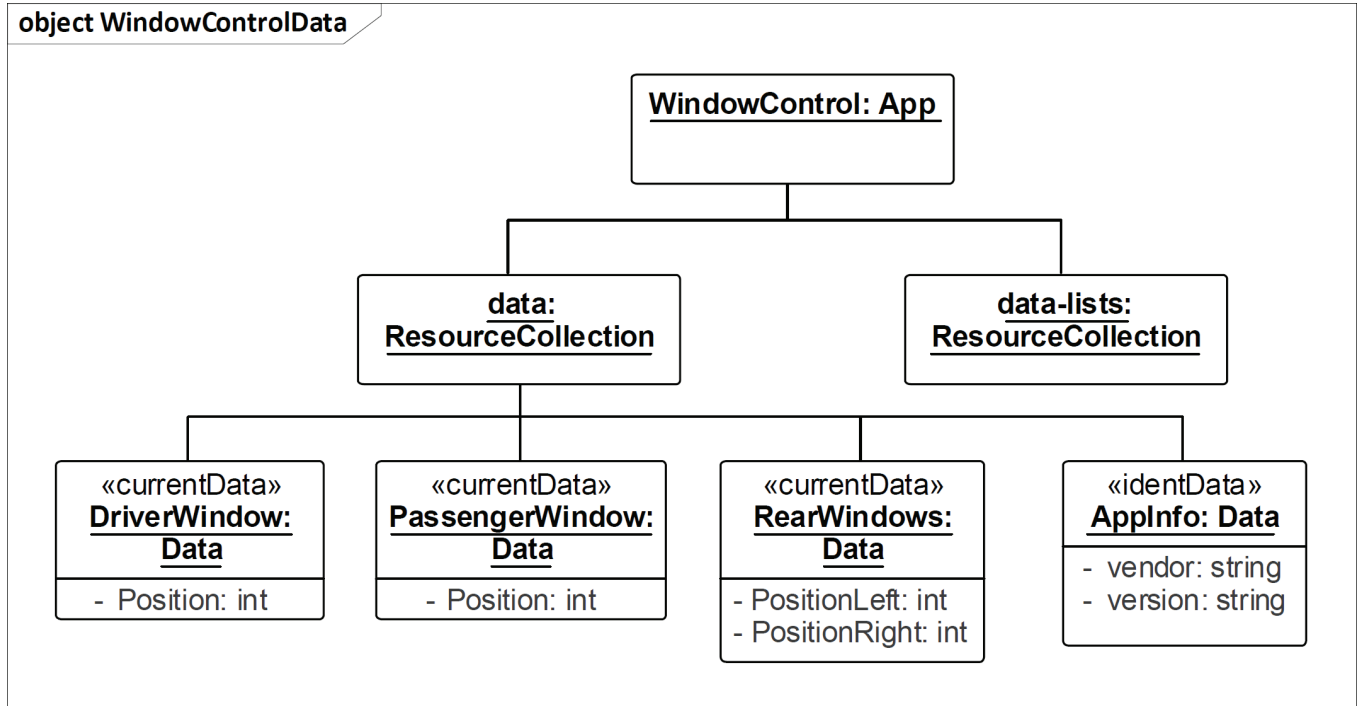


Figure 2 — WindowControl example

In terms of SOVD this is implemented by providing an SOVD **Entity** “WindowControl” containing the leaf resource “RearWindows”, which can be read using a **GET** operation.

The path elements “apps” and “data” of the resource path in the following example is explained in [5.3](#) and [5.4](#).

Example:

Request:

```
GET {base_uri}/apps/WindowControl/data/RearWindows HTTP/1.1
```

5.2.2 Server-Sent Events

For pushing content from an SOVD server to an SOVD client the SOVD API uses the Server-Sent Event mechanism as specified within the EventSource API ([\[31\]](#)).

The update of content is realized by an HTTP connection which is kept alive. The response is incrementally appended by new events, where each new event represents a new set of content. Examples for this mechanism are the cyclic reception ([7.10](#)) or the trigger-based reception ([7.11](#)) of data.

This section specifies the API method for retrieving content in increments using the Server-Sent Events mechanism. The setup of the URI, which provides the Server-Sent Events stream is covered in the sections of

the respective use cases. A server may detect lost clients via the Server-Sent Event stream. The handling of this scenario is also covered by the respective use cases.

Note [31] specifies the JavaScript API EventSource and the HTTP mechanism Server-Sent Events. For SOVD only the specification of Server-Sent Events shall be followed. While SOVD client and server shall adhere to the specification of Server-Sent Events, the Client API is an implementation detail not covered by this document.

5.2.2.1 Retrieving payload in increments

Method:

```
GET {event-stream-uri}
```

Method Description:

This method establishes a Server-Sent Event stream to enable pushing of payload from an SOVD server to an SOVD client.

Path Parameters:

[Table 3](#) specifies the path parameters to retrieve payload in increments.

Table 3 — Path parameters — Retrieving payload in increments

Parameter Name	Type	Convention	Description
event-stream-uri	string	M	URI of the Server-Sent Event stream.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use specific header parameters.

Response Headers:

```
Connection: keep-alive
Content-Type: text/event-stream
```

Response Status Codes:

[Table 4](#) specifies the response status codes to retrieve payload in increments.

Table 4 — Response status codes — Retrieving payload in increments

Status Code	Response Body	Description
200	—	The request was successful.

Response Body:

The response body shall follow the Server-Sent Event mechanism.([31]) Each event shall represent new content. Each event shall be transmitted using the EventEnvelope type. Only the event stream attribute data shall be used to transmit new content.

The SOVD attribute schema shall not be included in an event stream as it is provided upon creation of the subscription resource (e.g., cyclic subscription or trigger resource). The event stream attribute event shall not be used by SOVD as only one type of event is supported.

[Table 5](#) specifies the EventEnvelope type.

Table 5 — EventEnvelope type

Attribute	Type	Convention	Description
timestamp	string:date-time	M	Absolute timestamp when the value has been read by the SOVD server (UTC time).
payload	AnyValue	C ₁ ^a	The subject of the incremental update. This depends on the setup of the underlying use case (e.g., cyclic subscription or trigger resource)
error	GenericError	C ₂ ^b	The error returned by the underlying resource. The error shall be semantically equivalent to an atomic request on the resource. The attribute http_code inside parameters may be used to express the corresponding HTTP error if applicable. If an error has been returned, the client may still receive successful responses afterwards depending on the resource. Note: Partial errors (e.g., one value from a data response had error) are part of the payload as the resource's response was in general successful.

Example:**Request:**

```
GET {base_uri}/apps/WindowControl/cyclic-subscriptions/7ab52f10-5ea9-4610-alde-
f45ec8265ce4/events HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

data: {
data:  "timestamp":"2023-04-03T16:09:53Z",
data:  "payload": {
data:    "id": "RearWindows",
data:    "data": {
data:      "PositionLeft": 100,
data:      "PositionRight": 0
data:    }
data:  }
data: }

data: {
data:  "timestamp":"2023-04-03T16:10:23Z",
data:  "payload": {
data:    "id": "RearWindows",
data:    "data": {
data:      "PositionLeft": 75,
data:      "PositionRight": 50
data:    }
data:  }
data: }

data: ...
```

5.2.2.2 Unsubscribe from an event source

The client closes its connection to the event source by executing the close method on the EventSource object. Closing the connection does not make the server delete the subscription resource, as other clients are using the same subscription.

Example:

```
evtSource = new EventSource("{response.subscription}");

// Process responses
evtSource.onmessage = (event) => {
```

```

console.log(event.data.timestamp); // 2021-07-20T00:00:04.387819Z
payload = event.data.payload;
console.log(payload.id); // RearWindows
console.log(payload.data.PositionLeft); // 100
}

// Close event source
evtSource.close();

```

5.3 Entities

5.3.1 General

The SOVD API defines **Entities** holding diagnostic content. To access diagnostic content, an SOVD client shall specify the **Entity** it wants to interact with. This subclause covers **Entity** types, the hierarchy, and how to reference the **Entity** in a method request.

5.3.2 Entity types and collections

This document defines **Entity** types as shown in [Figure 3](#). **Entity** types are defined flexibly enough to support various topologies and diagnostic strategies. **Entities** of same type are grouped within **Entity** collections under the SOVD server. SOVD **Entities** hold the diagnostic content as resources. Resource collections available for an **Entity** type are defined in [5.4.2](#).

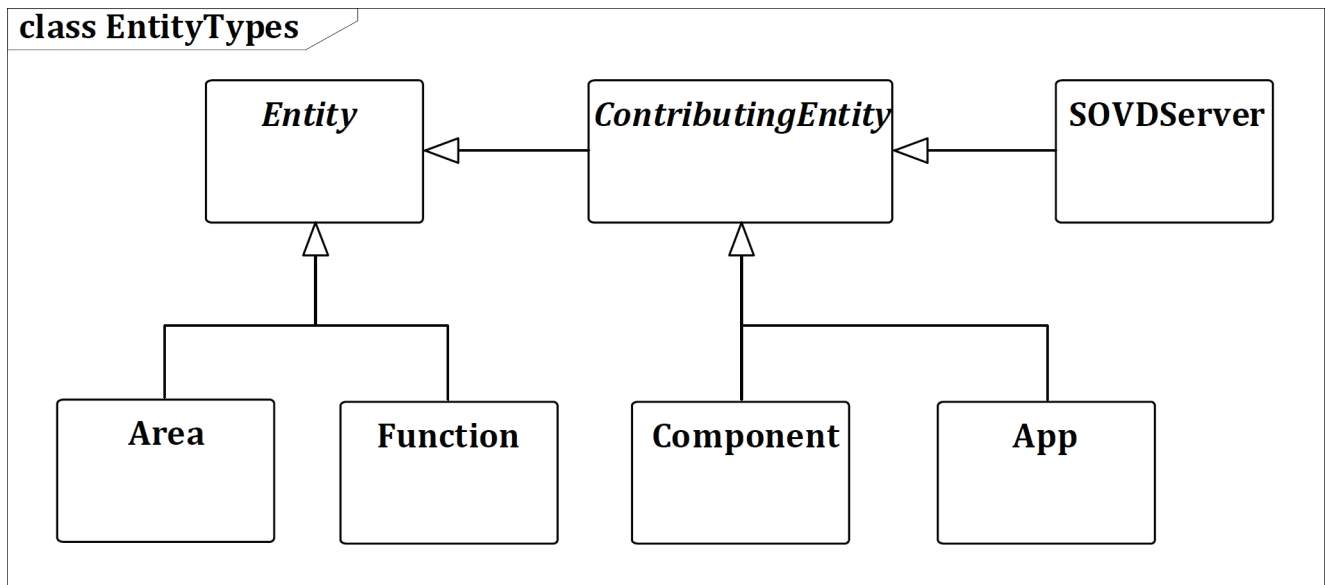


Figure 3 — Entity types overview

[Table 6](#) specifies the different **Entity** types in more detail.

Table 6 — Details of Entity types

EntityType	EntityCollectionType	Description
SOVDServer	—	An SOVDServer represents the entry point of the API structure. Resources on that level are related to all Entities managed by the SOVD server.
Area	areas	An Area represents a logical view and can be used to describe various vehicle topologies, e.g., domain architectures (body control, infotainment, powertrain), zone architectures (front, rear), or logical architectures in the context of a software defined vehicle.
Component	components	A Component is a representation of hardware or software. A Component can also represent a combination of both (e.g., ECU with basic software). Components representing hardware are characterized by the availability of a CPU and execute software. Examples are HPCs, ECUs, and Smart sensors. Software as part of Components is characterized as basic software required to execute Apps. Examples for basic software are operating systems, or Hypervisor. Only software Components or Components containing software parts shall link to Apps.
App	apps	An App represents an Application executed on a Component , e.g., Advanced-Lane-Keeping or Window Control.
Function	functions	A Function represents a functional view and allows an ExVe manufacturer to define access to diagnostic information, which can be spread across several Components. Note: This document defines no further details for the handling of Functions. The REST approach limits access to a single resource per request. To accumulate values from multiple Entities with a single requests Functions may be used. Annex B.5 shows an example of a Function Entity “Vehicle-Health” to retrieve a list of faults from multiple Components along with HW and SW version numbers.

From an API perspective, there is no differentiation between hardware and software **Components**.

5.3.3 Entity hierarchy and relations

The **Entity** defined can be used to build a hierarchy for the SOVD API. [Figure 4](#) depicts the standardized **Entity** hierarchy. The entry point into this structure is the SOVD server. For simplicity of [Figure 4](#), the {entity-collection}s are not shown as separate classes but can be seen from the name of the respective association ends.

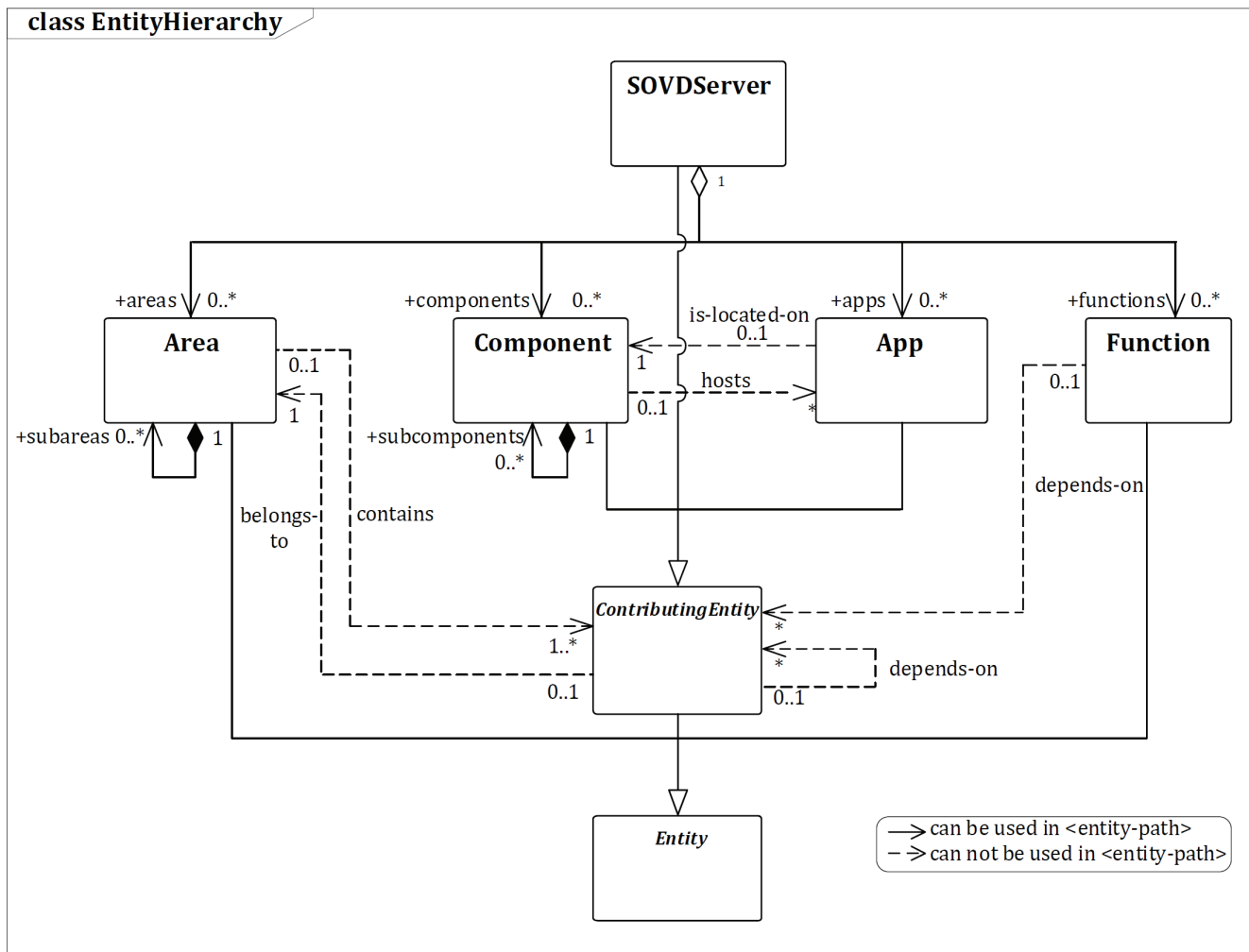


Figure 4 — Entity hierarchy

A **Component** can have multiple subcomponents, which themselves are **Components** and thus can have subcomponents. **Components** on all levels can have links to **Apps** (identified by the relation *hosts*). This information can be used by an SOVD client to present all **Apps** running on a **Component**. An **App** itself can provide the information about which **Component** it is located on using the relation *is-located-on*.

Some SOVD **Entities** may depend on other **Entities**, e.g., an **Application** may depend on information provided by another **Application**. By introducing the abstract parent class *ContributingEntity* as well as the relationship *depends-on*, this dependency can be modelled for all **Entities** extending *ContributingEntity*, namely **Entities** of type **Component**, **App**, and **Function**.

Using the dependency *contains* of an **Area** to a *ContributingEntity* both physical and logical hierarchies can be represented. The same applies for the opposite possibility of a *ContributingEntity* to provide the **Area** it *belongs-to*.

An **Area** can have multiple subareas, which themselves are **Areas** and thus can have subareas.

In addition, the *depends-on* relation from a **Function** to a *ContributingEntity* can be used to represent which **Entities** contribute to a certain **Function**.

5.3.4 Entity identifier and path

An **Entity** identifier uniquely identifies an **Entity** for a given SOVD server on a single level in the hierarchy. A sub-entity may have the same identifier as the parent **Entity**, as they reside on different hierarchy levels. This document does not define **Entity** identifiers.

Every **Entity** is accessible via its `entity-path`. The **Entity** path is built from the element's `EntityCollectionType` (see [Table 6](#)), and the **Entity** identifier.

The path elements are separated by a forward-slash ('/'). In case of a sub level **Entity** all path elements are concatenated.

Example for entity-path:

```
components/DrivingComputer
components/DrivingComputer/subcomponents/Linux
```

5.3.5 Location of the SOVD server

The location of the SOVD server is defined by the vehicle manufacturer. It can be located inside the vehicle as well as outside the vehicle to provide an SOVD interface also for legacy vehicles.

5.3.6 Multi-SOVD server environments

5.3.6.1 General

This document neither defines nor demands a specific architecture within an ExVe. It is intended to support multiple kinds of architectures, including multiple SOVD servers for a single ExVe.

It is strongly recommended, that each **Entity** for an ExVe is bound to and can only be accessed through a dedicated SOVD server (see [5.3.2](#)). An SOVD server can be public, meaning accessible from outside or private, meaning it is not accessible from the outside

5.3.6.2 Public SOVD server

Each public SOVD server is directly accessible from the outside through connectivity. It is outside the scope of this document to provide identification of a public SOVD server for remote SOVD clients. For co-located clients, the identification method as described in [5.11](#) shall be used. [Figure 5](#) shows an example with three public SOVDServer.

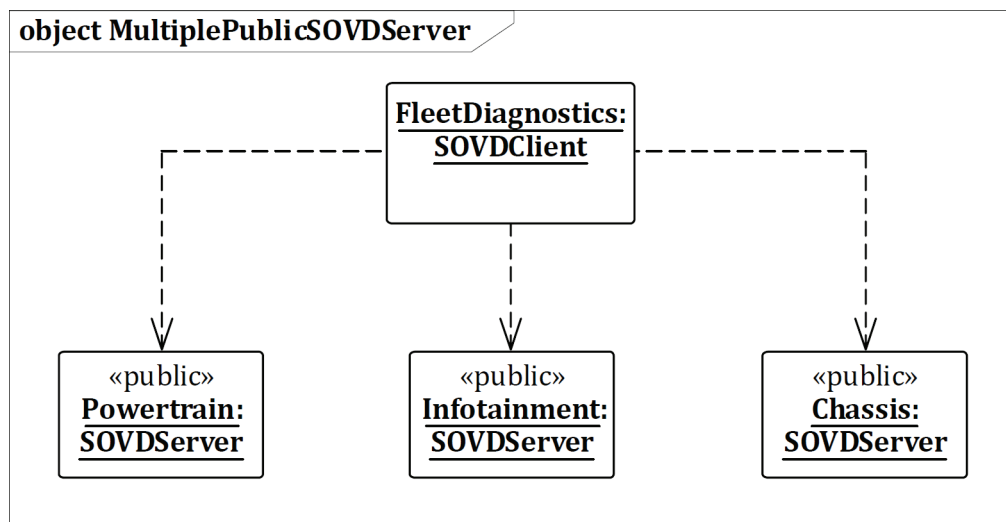


Figure 5 — Public SOVD server

5.3.6.3 Private SOVD server

An SOVD server can be restricted to internal ExVe access only, allowing direct access solely for an SOVD client within ExVe services (including from legacy ECUs via a CDA) which are provided by a private SOVD server which are intended to be available outside the ExVe have to be forwarded to a public SOVD server. [Figure 6](#) shows an example with one public and one private SOVDServer.

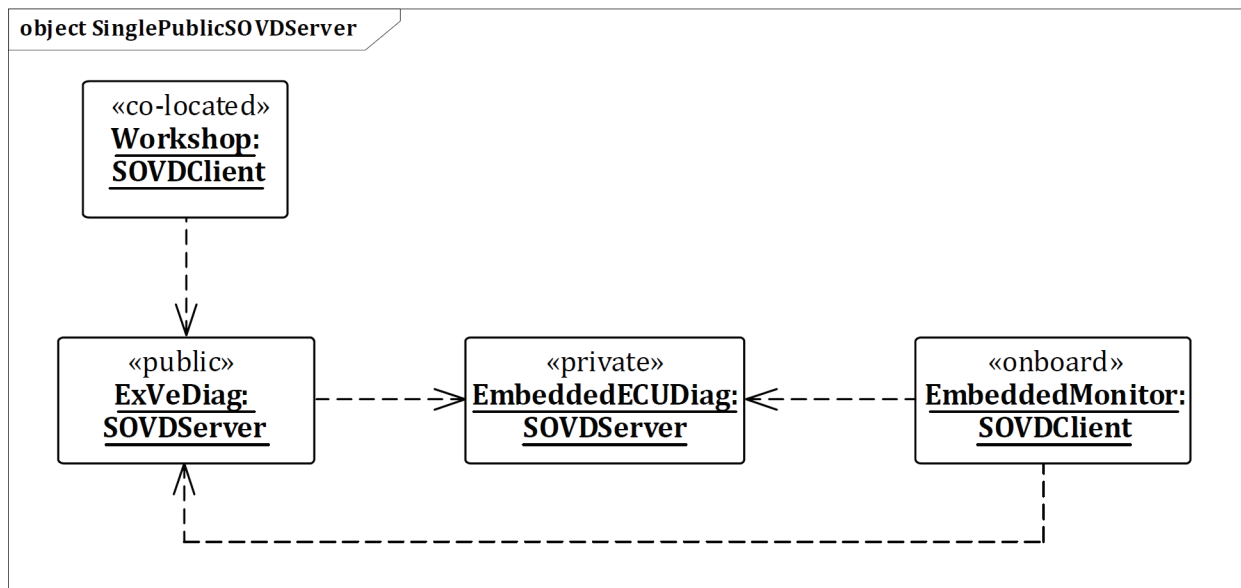


Figure 6 — Private SOVD server

5.4 Resources

5.4.1 General

Each **Entity** provides its diagnostic content in the form of resources at the SOVD API. These resources are accessible either at entity-level or via their respective resource collections depending on the underlying type of diagnostic content they provide.

Appendix [B.3.3](#) shows an example for resource collection data containing resources of type data.

This clause describes the standardized resource types, how they are organized into collections, and which are available for a specific **Entity** type.

Constraints:

- A resource shall only be accessible via a single path.
- Access shall only be provided to an entire resource. It shall not be possible to access parts of a resource by means of filters, e.g., a single parameter of a data or configuration resource.
- A resource shall only be available on the **Entity** that represents the diagnostic content. An **Entity** shall not aggregate resources which are held by e.g., subcomponents.

5.4.2 Standardized resource collections

[Table 7](#) specifies the standardized resource collections for SOVD. This resource collection classification helps a generic tester to present the resources of different types in a specific way. Each resource collection can be queried for its contained resources.

Table 7 — Standardized resource collections

Resource Collection	Description
configurations	Collection of configuration resources
bulk-data	Collection of bulk data resources for the respective BulkDataCategory
communication-logs	Collection of communication logs
cyclic-subscriptions	Collection of cyclic subscriptions
data	Collection of static and dynamic data resources
data-lists	Collection of combined data resources
faults	Collection of fault resources
operations	Collection of operation resources
updates	Collection of update package resources
modes	Collection of mode resources
locks	Collection of lock resources
triggers	Collection of trigger resources
scripts	Collection of script resources

Table 8 gives an overview on which resource collections are allowed on which **Entity** type.

Table 8 — Relation between Entities and resource collections

Entity level	bulk-data	communi- cation-logs	cyclic-sub- scriptions	configura- tions	data	data-lists	faults	locks	modes	operations	scripts	triggers	updates
SOVDServer	X	X	X	X ^a	X ^a	X ^a	X ^a	X	X	X	X	X	X
Area	not supported by this document												
Component	X	X	X	X	X	X	X	X	X	X	X	X	X
App	X	X	X	X	X	X	X	X	X	X	X	X	X
Function	not supported by this document												
^a It is strongly recommended that resource collections on an SOVDServer level do not aggregate resources from entities underneath.													

5.4.3 Standardized resources

Table 9 defines the resource names for an **Entity**.

Table 9 — Standardized resource names

Name	Description
docs	Provides the online capability description.
version-info	Provides supported SOVD API versions.
logs	Access to logging information
related-apps	DEPRECATED Apps hosted on a Component
related-components	DEPRECATED Components related to an Area
data-categories	Categories (currentData, identData, ...) used within data resource collections.
data-groups	Groups used to structure data in a specific way (e.g., engine performance related, ...)
authorize	Initiate authentication and authorization with the vehicle

Table 9 (continued)

Name	Description
token	Request token from the vehicle for authorization
is-located-on	Provides the information that an App is located on a certain Component
hosts	Provides the information about which Apps are hosted on a certain Component
contains	Provides the information about which ContributingEntities are contained in an Area
belongs-to	Provides the information about which Area a ContributingEntities belongs to
depends-on	Provides information about ContributingEntities on which a ContributingEntity depends on
clear-data	Clears temporary data of an Entity.

[Table 10](#) gives an overview, which resource is allowed to be used on which **Entity** type.

Table 10 — Relation between Entities and resources

EntityType	docs	version-in-fo	logs	is-located-on	hosts	contains	belongs-to	depends-on	data-categories	data-groups	authorize	token
SOVDServer	X	X	X	— ^a	— ^a	— ^a	X	X	X	X	X	X
Area	X	— ^a	— ^b	— ^a	— ^a	X	— ^a	— ^a	— ^b	— ^b	— ^b	— ^b
Component	X	— ^a	X	— ^a	X	— ^a	X	X	X	X	— ^b	— ^b
App	X	— ^a	X	X	— ^a	— ^a	X	X	X	X	— ^b	— ^b
Function	X	— ^a	— ^b	— ^a	— ^a	— ^a	— ^a	X	— ^b	— ^b	— ^b	— ^b
^a not allowed												
^b not supported by this document												

Resources, which are created by an SOVD client by usage of the SOVD API are considered “temporary”. Consequently, they are not available in a capability description.

Note An example of resources and resource collections is shown in [Annex B](#).

5.4.4 Restricting access to entities and resources by authentication

The SOVD server shall ensure the protection of all entities and resources against unauthorized access. This standard does not define if a client is eligible to connect to the SOVD server, how an SOVD client must perform an authorization, or which concept is used to define which client can access which entity or resource. [Section 7.23](#) informally describes an implementation based on OAuth and TLS.

Entities and resources may be accessible without prior authorization, e.g., a server may allow any eligible client to read fault codes and identifications. For all entities, resources, and methods which require authorization, the SOVD server shall only grant access client has been both authenticated and authorized. For SOVD methods that enumerate resources (e.g., data-lists, subcomponents, ...) or event sources that send resource content (e.g. cyclic subscriptions, triggers), the server shall ensure that the client does not receive information about resources they are not authorized to access. In such cases, the server may filter out the inaccessible resources. For temporary resources, the SOVD server defines if the access is restricted to the client which created the temporary resource or if other clients can access the temporary resource as well or if the authorization requirements from the parent resource are inherited.

5.4.5 Custom specific resource collections and types

Additional custom resource types, collections, and top-level-resources may be defined. The naming convention `x-<ext>-<name>` shall be followed for these custom extensions. Thereby “<ext>” shall be a unique identifier that leads back to the source of the custom extension to avoid conflicts with other potential sources. The placeholder “<name>” shall be a unique identifier for the resource type or the collection type

within the namespace defined by “<ext>”. “x-sovd-” shall never be used for this purpose and is reserved to this document.

Note Since these resource collections and types are ExVe manufacturer specific, it cannot be assumed that a generic SOVD client can automatically support this.

Example:

ExVe manufacturer “*Cop Cars*” wants to introduce a new resource type “*special*”, which is accessible via a new resource collection. To avoid a naming conflict with ExVe manufacturer “*Spy Vehicles*”, who also introduced a resource type “*special*”, the resource collection for ExVe manufacturer “*Cop Cars*” could be named `x-cop-cars-specials` and the resource type `x-cop-cars-special`. In this example, “cop-cars” replaces “ext” and “specials” or “special” represents the “name”.

5.5 Illustration of SOVD concepts for Entities and resources

[Figure 7](#) schematically shows the structure of an **SOVDServer** which contains two **Areas**, three **Components** and one **App** as **Entities**. For **Component 3** and **App 1** [Figure 7](#) also shows the resource collections and accessible resources.

Note To keep [Figure 7](#) simple, not all relations as defined by this document are displayed.

Solid lines are used to illustrate the single path for accessing specific information. The dashed lines show related **Entities**.

For resources data1, data2 and operation1 the URI for accessing them is provided.

Resources are located below **Entities**. To better distinguish between **Entities** and resources, resources are represented by boxes with rounded corners.

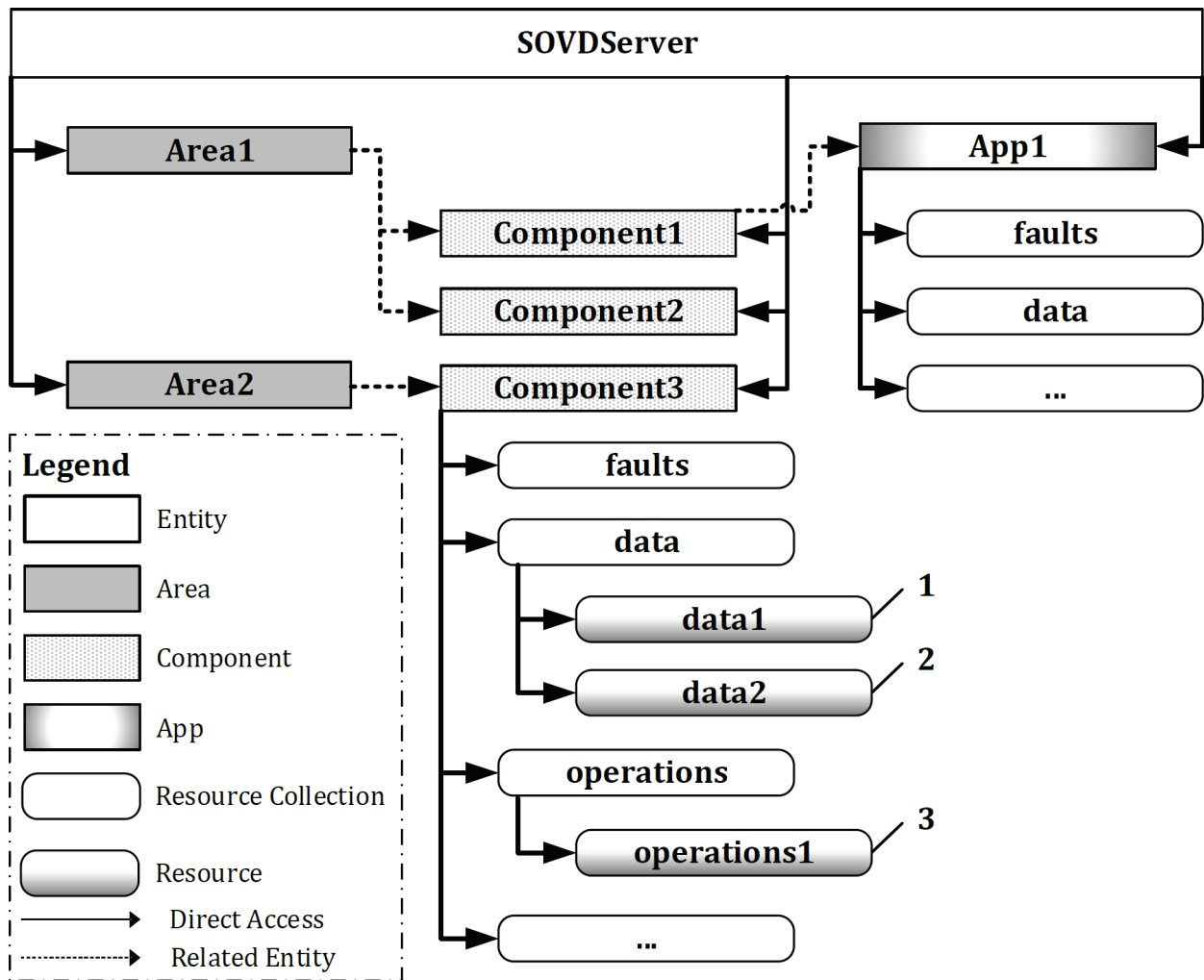


Figure 7 — Simplified illustration of the relationships between Entities and resources

Note An example of **Entities**, resources, resource collections is shown in [Annex B](#).

5.6 SOVD API versioning

The SOVD API uses URI based versioning. Thus, the version of the SOVD API is included in the URI. For this document base_uri shall be defined as:

`https://<sovd-server-host>[/<exve-manufacturer-specific>]/<version-identifier>.`

The mandatory path element <sovd-server-host> shall express the host name as per RFC6763.([19]) The mandatory path element <version-identifier> shall have the value “v1” for this version of the document. The optional path element <exve-manufacturer-specific> can be used to avoid clashes in the URIs provided by ExVe HTTP servers.

If an SOVD client uses a version not supported by the SOVD server, the server shall respond with the status code 404 (not found). An SOVD server may support multiple versions or may redirect an SOVD client to the same URI of the supported version if this URI provides syntactically the same information.

The SOVD API shall provide the `version-info` resource to allow SOVD clients to read all supported versions of the SOVD server. The path to the `version-info` resource shall remain the same for this and all future versions of the SOVD API.

5.7 Internationalization and localization

5.7.1 Internationalization

Most resources accessible through the SOVD API provide a human-readable name using the attribute name. This name can be translated using the identifier defined by the attribute `translation_id` which is provided as a sibling of the attribute `name`.

If the response contains multiple internationalizable texts (see [Table 61](#)), the translation identifier is built up by using the name of the internationalizable attribute and appending `_translation_id`. For example, in [Table 61](#) the attribute `symptom` can be internationalized and thus the internationalization identifier is provided by the attribute `symptom_translation_id`.

5.7.2 Unit definitions

The SOVD API provides information on the unit of a value (e.g., odometer distance in kilometers) so that the data can be interpreted correctly. For converting the value into different unit systems (e.g., imperial and metric), the unit information provided by the SOVD API contains the physical dimensions as well.

Basis for the conversion of a derived quantity are the base SI units. A derived quantity is converted from a quantity in base units by the equation:

$$value_{derived} = F(value_{base}) = factor_{si_to_unit} * value_{base} + offset_{si_to_unit}$$

The unit information is provided as part of the schema in the attribute `x-sovd-unit` with the following attributes:

[Table 11](#) specifies the `SovdUnitType`.

Table 11 — SovdUnitType

Attribute	Type	Convention	Description
<code>display_name</code>	string	M	The string for displaying the unit in an SOVD client (e.g., “km/h” or “min ⁻¹ ”.
<code>reference</code>	string	0	Identifier for the unit for localization, e.g., as provided by the ODX_RS_UNIT_LIB.odx-d [21].
<code>factor_si_to_unit</code>	number	0	Factor to calculate the value in the defined unit. Default value is 1.
<code>offset_si_to_unit</code>	number	0	Offset to add to the value. Default value is 0. Note: For a differential value, e.g., temperature difference, this attribute should be omitted.
<code>physical_dimension</code>	PhysicalDimension	C ^a	Definition of the physical dimension based on SI units.
^a C: unit has physical dimension.			

[Table 12](#) specifies the `PhysicalDimension` type.

Table 12 — PhysicalDimension type

Attribute	Type	Convention	Description
id	string	O	Identifier for the physical dimension
length	integer	C ^a	Exponent for length
mass	integer	C ^a	Exponent for mass
time	integer	C ^a	Exponent for time
current	integer	C ^a	Exponent for current
temperature	integer	C ^a	Exponent for temperature
molar_amount	integer	C ^a	Exponent for molar-amount
luminous_intensity	integer	C ^a	Exponent for luminous intensity
plane_angle	integer	C ^a	Exponent for plane angle
^a C: only set if dimension is not 0.			

Example:

```
{
  "x-sovd-unit": {
    "display_name": "km/h",
    "reference": "speed_in_km_per_h",
    "factor_si_to_unit": 3.6,
    "offset_si_to_unit": 0,
    "physical_dimension": {
      "id": "speed",
      "length": 1,
      "time": -1
    }
  }
}
```

5.7.3 Internationalization of enumerations

The SOVD API defines a couple of enumerations as follows:

```
"EnumDefinition": {
  "type": "string",
  "enum": [
    "value1",
    "value2",
    "value3"
  ]
}
```

To enable SOVD clients to display the values of an enumeration in other languages, this document defines the custom extension `x-sovd-enum-translation-ids` (see [Table 25](#)). It basically contains key-value pairs where the key is the enumeration value (e.g., `value1`) and the value contains the translation identifier (e.g., `text-id1`). If required, the attribute `x-sovd-enum-translation-ids` is contained in the definition of the enumeration as shown below.

```
"EnumDefinition": {
  "type": "string",
  "enum": [
    "value1",
    "value2",
    "value3"
  ],
  "x-sovd-enum-translation-ids": {
    "value1": "text-id1",
    "value2": "text-id2",
    "value3": "text-id3"
  }
}
```

The schema for `x-sovd-enum-translation-ids` is defined as follows:

```
"x-sovd-enum-translation-ids": {
  "type": "object",
```

```

    "additionalProperties": {
      "type": "string"
    }
  }
}

```

5.7.4 Other localization topics (numbers and dates)

Numbers are always transmitted using JSON numbers. Thus, there is no need to specify a localization for numbers.

Dates provided by the SOVD server (e.g., expiration of locks) are always provided according to RFC 3339 [15] and require no localization on the SOVD server side.

If an **Entity** provides dates (e.g., the programming date of an **Entity** provided as a data resource), the SOVD server shall keep the existing format and not perform localization.

5.8 Response status codes

5.8.1 General

The SOVD API aims to follow the standard HTTP status codes as defined in RFC 7231 ([2] chapter 6), to indicate the success or failure of a request. The following status codes are defined:

- 2xx success status codes confirm the successful execution of the request.
- 4xx error status codes indicate an error on the SOVD client side.
- 5xx error status codes indicate an error on the SOVD server side.

5.8.2 Generic definition of status codes

In the following tables the commonly used response status codes which are applicable for the SOVD API are described. The description of a specific method may provide additional details or further response status codes if necessary.

[Table 13](#) specifies the common SOVD server-side supported status codes on success.

Table 13 — Common SOVD server-side supported status codes on success

Code	Text	Description
200	OK	The request has been successful. This response code is also used if empty collections are returned.
201	Created	A new resource, e.g., a data-list, has been created.
202	Accepted	The request has been accepted, but a response is not yet available. More details on the response are defined in the methods which return this status code.
204	No Content	The request has been successful, but the response provides no processable content.

[Table 14](#) specifies the common SOVD server-side supported status codes on SOVD client error.

Table 14 — Common SOVD server-side supported status codes on SOVD client error

Code	Text	Description
400	Bad request	The request from the SOVD client was invalid.
401	Unauthorized	The access to the requested resource requires prior authentication. No or insufficient authentication information has been provided (expired, malformed, incorrect credentials, ...).
404	Not found	The requested resource could not be found.
405	Method not allowed	The SOVD server knows the request method, but the target resource doesn't support this method. This especially applies to resources which cannot be deleted by the client, regardless of access rights.
406	Not acceptable	The requested resource representation is not supported.
409	Conflicted	The request conflicted with the current state of the Entity or resource.
415	Unsupported Media Type	The request could not be serviced because the payload is in a format not supported by the method of the target resource.

[Table 15](#) specifies the common SOVD server-side supported status codes on SOVD server error.

Table 15 — Common SOVD server-side supported status codes on SOVD server error

Code	Text	Description
500	Internal Error	An internal error occurred.
501	Not implemented	The SOVD server does not support the required functionality.
503	Method Unavailable	The method is temporarily unavailable. Optionally, the SOVD server can use the Retry-After header to tell the client the time after which it should retry execution of method.
504	Gateway Timeout	No response received from the underlying Entity (e.g., no UDS response from an ECU) in a given amount of time.

5.8.3 Definition of error details

It is recommended that the response body of an error response includes a more detailed description of the error which helps in diagnosing the issue. The method descriptions in [Clause 7](#) use the GenericError type as specified in [Table 16](#) to provide additional information.

Table 16 — GenericError type

Attribute	Type	Convention	Description
error_code	string (see ErrorCode)	M	SOVD standardized error code
vendor_code	string	C ^a	Vendor specific error code
message	string	M	Message describing the problem in more detail.
translation_id	string	O	Identifier for translating the message
parameters	Map of string values (defined as additionalProperties)	O	Key-value-list which provides additional information on the error.

^a C: if the value of error_code is vendor-specific this attribute shall be filled.

On some occasions an error applies only to parts of a response, e.g., when reading a data resource. In this case the DataError type, as specified in [Table 17](#) is used, which includes a path to the erroneous attribute of the response:

Table 17 — DataError type

Attribute	Type	Convention	Description
path	string:json-pointer	M	A JSON Pointer describing which element of the response is erroneous.
error	GenericError	C ^a	Describes the error more precisely.
^a C: the attribute referenced by path contains an error.			

5.8.4 SOVD error codes for common cases

[Table 18](#) describes the SOVD error codes for common cases.

Table 18 — SOVD error codes for common cases

ErrorCode	Description
error-response	The Component receiving the request has answered with an error. The message shall include relevant information to properly identify the errors (for UDS the message should include the service identifier (Key: 'service' and Value of type <code>number</code>) and the negative response code (Key: 'nrc' and Value of type <code>number</code>)).
incomplete-request	The request does not provide all information (e.g., parameter values for an operation) required to complete the method. The message should include references to the missing information.
insufficient-access-rights	The SOVD client does not have the right to access the resource.
invalid-response-content	The response provided by the Component contains information which could not be processed. E.g., the response of an ECU does not match the conversion information known to the SOVD server. The message should include references to the parts of the invalid response attribute as well as a reason why the attribute is invalid.
invalid-signature	The signature of the data in the payload is invalid.
lock-broken	The lock of the client has been broken by another client. In this case the SOVD server sends back the status code 409 (Conflicted).
not-responding	The Component which handles the request (e.g., an ECU) has been queried by the SOVD server but did not respond.
precondition-not-fulfilled	The preconditions to execute the method are not fulfilled.
sovd-server-failure	The SOVD server is able to answer requests, but an internal error occurred. The message should include further information about the error.
sovd-server-misconfigured	The SOVD server is not configured correctly, e.g., required configuration files or other data is missing. The message should include further information about the error. A client shall assume that this error is fatal, and regular operation of the SOVD server cannot be expected.
update-automated-not-supported	Another update is currently being executed and has not yet finished or aborted.
update-execution-in-progress	Another update is currently executed and not yet done or aborted.
update-preparation-in-progress	An update is already in preparation and not yet done or aborted.
update-process-in-progress	An update is already in progress and not yet done or aborted.
vendor-specific	Details are specified in the <code>vendor_code</code>

5.9 Common primitive data types

[Table 19](#) presents the list of common JSON data types used within the SOVD API in addition to those defined in the OpenAPI specification (see [\[7\]](#) which itself is defined on top of the JSON value types introduced in RFC 8259 (see [\[23\]](#))).

Note In this specification the format **x:y** is used in the definition for an attribute's type to denote that x refers to the JSON type and y provides a basic semantic identification of the value following the JSON Schema validation.

Table 19 — SOVD primitive data types

JSON Type	JSON Format	Comment
string	byte	Base64 encoded characters
string	binary	A sequence of octets, e.g., binary/bulk data
string	password	A hint to UIs to obscure input.
string	email	An email address according to RFC 5322 (see [24])
string	uri	An absolute URI according to RFC 3986 (see [25]), e.g., {base_uri}/components/ecu-1
string	uri-reference	A URI reference according to RFC 3986 (see [25]), e.g., /components/ecu-1
string	uri-template	A URI template string according to RFC 6570 (see [26]), e.g., /components/{ecu-id}/data
string	json-pointer	A string pointing to a specific value within a JSON document according to RFC 6901 (see [27]), e.g., #/data/batteryVoltage
string	uuid	A Universally Unique Identifier (UUID) according to RFC 4122 (see [28])
string	date	A full date according to RFC 3339 (see [15])
string	date-time	A date-time according to RFC 3339 (see [15])
string	time	A time according to RFC 3339 (see [15])
string	duration	A duration according to RFC 3339 (see [15])
string	period	A period according to RFC 3339 (see [15])
string	regex	Regular expressions as defined in ECMA 262 (see [14])
string	hex	A string representing a hexadecimal number, e.g., "fe001a34"
AnyValue	—	The type of the attribute is either object, array, number, integer, boolean or string.

5.10 Definition of JSON schema and objects for vehicle responses

The SOVD API returns information retrieved from the vehicle as JSON objects and utilizes the JSON schema for representing the schema (structure, attributes, types, etc.) of the object. A common example is that the vehicle returns the programming and coding date of an ECU as a response structure like shown in [Figure 8](#).

PA_Programming_Date	
	PA_Year: A_UINT32 = 2021
	PA_Month: A_UINT32 = 10
	PA_Day: A_UINT32 = 10
PA_Coding_Date	
	PA_Year: A_UINT32 = 2021
	PA_Month: A_UINT32 = 10
	PA_Day: A_UINT32 = 10

Figure 8 — Example response structure

For this response structure the following example JSON schema and object is generated as shown in [Table 20](#).

Table 20 — Mapping of example response structure to JSON

Schema	Object
<pre> { "type": "object", "properties": { "PA_Programming_Date": { "type": "object", "properties": { "PA_Year": { "type": "integer" }, "PA_Month": { "type": "integer" }, "PA_Day": { "type": "integer" } } }, "PA_Coding_Date": { "type": "object", "properties": { "PA_Year": { "type": "integer" }, "PA_Month": { "type": "integer" }, "PA_Day": { "type": "integer" } } } } } </pre>	<pre> { "PA_Programming_Date": { "PA_Year": 2021, "PA_Month": 10, "PA_Day": 10 }, "PA_Coding_Date": { "PA_Year": 2021, "PA_Month": 10, "PA_Day": 10 } } </pre>

The rules for generating the JSON schema and object are as follows:

- Simple attributes are mapped to JSON attributes using a type as defined in Common primitive data types.
- Structures are represented as a JSON object where each of the attributes of the structure is an attribute in the JSON object. For the enclosed attributes the mapping described here is applied again.
- Arrays are mapped to JSON arrays. The element's type in the array is determined by the type of the JSON element in the JSON array. For example, an array of strings would be mapped to a JSON array of JSON strings, and an array of complex structures would be mapped to a JSON array of JSON objects.
- Multiplexers are complex structures where the relevant attributes are defined by a value. The attribute which defines the type to be used is specified using the `Discriminator` object as defined by the OpenAPI specification together with `oneOf` definition in the JSON schema to define the JSON schema for the possible values.

5.11 Discovery of the SOVD server for the co-located use case cluster

5.11.1 General concept

Use Multicast DNS (mDNS, RFC6762 [16]) to obtain the IP address and DNS Service Discovery (DNS-SD, RFC 6763 [19]) to identify SOVD servers. It is recommended to define a suitable Time To Live (TTL) for mDNS to ensure that Services which have disappeared from the network do not appear to be valid for a longer period.

5.11.2 Network connection

There might be the need for additional vehicle-specific steps to activate the mDNS and DNS-SD server.

The SOVD server shall connect to the local network using an existing DHCP server or if not available use an AutoIP address.

The SOVD server shall listen on the port 7690 (registered at IANA [43]).

5.11.3 Address an SOVD server using mDNS

The public SOVD server shall implement mDNS host according to RFC6762.([16]) It shall respond to its local hostname which consists of the hostname (could be the VIN or user defined) (e.g., "ABC123456789.local").

In case of multiple public SOVD servers, each public SOVD server shall be assigned a unique hostname (e.g., by extending the VIN with (1), (2) and so on).

Note 1 This is already done by the mDNS protocol: In case a foreseen hostname already exists in the subnet, e.g., "(1)" is added to the hostname.

Note 2 It is recommended for each SOVD server to provide its own unique hostname in order not to have non-deterministic name assignment (with (1), (2), etc.).

5.11.4 Discovery of the SOVD server using DNS-SD

The public SOVD server shall implement DNS based service discovery as described in RFC6763 ([19]) with the following parameters:

- Protocol description: ASAM SOVD
- Primary Transport Protocol: TCP
- TXT Record key-value pairs
- identification: Identification Information of the vehicle, e.g., the vehicle identification number
- accessurl: URI for accessing the version-info resource of the SOVD server

The `identification` parameter shall contain a unique identification of the SOVD server, such as the VIN of the vehicle.

The `accessurl` parameter shall point the client to the root of the SOVD API which is the parent of the `version-info` resource and the `version` identifier to form the `base_uri`. With regard to 5.6 the `accessurl` is defined as `https://<SOVD-Server-Host> [<exve-manufacturer-specific>]`.

After discovering the SOVD server via DNS-SD, the SOVD client can use the `accessurl` to query the `version-info` resource and determine which SOVD version(s) the server supports (`GET (<accessurl>/version-info)`). If the client is looking for a specific SOVD version it can directly try to access resources at the version specific `base_uri` (e.g., `<accessurl>/v1/...`).

Example:

```
identification=ABC122345789
accessurl=https://ABC122345789.local:7690/vehicle/version-info
```

5.11.5 Example

Client asks via DNS-SD for available SOVD services (broadcast):

```
$ dns-sd -B _sovd._tcp local.
```

Answer:

```
Timestamp A/R Flags if Domain Service Type Instance Name
23:54:47.783 Add 3 4 local. _sovd._tcp. ABC123456789
```

Then the client asks the vehicle for infos (lookup):

```
$ dns-sd -L "ABC123456789" _sovd._tcp. local.
```

Answer:

```
DATE: ---Thu 02 Dec 2021---
16:44:24.438 ...STARTING...
16:44:24.816 ABC123456789._sovd._tcp.local. can be reached at
ABC123456789.local.:7690 (interface 11)
```

Then the IP address is resolved:

```
$ dns-sd -Gv4v6 ABC123456789.local
```

Answer:

```
DATE: ---Thu 02 Dec 2021---
16:48:50.274 ...STARTING...
Timestamp A/R Flags if Hostname Address TTL
16:48:50.257 Add 40000003 11 ABC123456789.local.
2A02:8070:A1C2:...8EFF:FE35 120
16:48:50.257 Add 40000002 11 ABC123456789.local. 192.168.178.28 120
```

5.12 Discovery of the SOVD server for the remote use case cluster

In case the SOVD API is also provided for the remote use case cluster, there are several ways how to discover and how to address individual vehicles. The appropriate approach is not specified in this document.

6 Capability description

6.1 General

This description introduces the structure and definitions underlying the SOVD capability descriptions. The documentation of an SOVD API exposed by an SOVD Server, in form of a SOVD capability description, builds on top of the OpenAPI Specification [\[2\]](#) (with version $\geq 3.1.0$).

For the convenience of the user, a machine-readable definition of the API methods (YAML) is published alongside this document. It is recommended to use the structure and generic types described in this file when building an offline capability description.

6.2 Capability description structure

6.2.1 General

An SOVD capability description is written in the form of the OpenAPI Specification. [Table 21](#) lists the mandatory fields of an OpenAPI Specification which shall be used for documenting an SOVD API, if the respective feature is implemented by the underlying SOVD Server.

Table 21 — SOVD API required fields

Attribute	Convention	Description
servers	M	Documentation of the single SOVD Server the SOVD capability description is defined for.
paths	M	Documentation of SOVD resources implemented by the SOVD server.
components	M	Documentation of the SOVD-specific definitions as well as ExVe manufacturer specific reusable definitions, e.g., parameters, responses, schemas, etc.
security	M	Documentation of the API security mechanism implemented by the SOVD Server.

The following list of fixed fields introduced by OpenAPI Specification are not used as part of an SOVD capability description: *jsonSchemaDialect*, *webhooks*, and *externalDocs*.

Further SOVD-specific extension properties are introduced in the following subclauses using the underlying extension mechanism of the OpenAPI Specification. These extension properties allow to document further SOVD specific information and are therefore grouped based on the OpenAPI Specification objects they can be specified for. For example, the SOVD Standard version underlying to an SOVD capability description can be documented as part of the `Info` object of an OpenAPI document as presented in [6.2.2](#).

6.2.2 Info object

[Table 22](#) provides an overview of all SOVD extension properties for an OpenAPI Info object.

Table 22 — Extension Property Info

Attribute	Type	Convention	Description
x-sovd-version	string	M	Documents the SOVD standard version underling to the capability description.
x-sovd-tags	string[]	C ^a	Documents the SOVD <code>tags</code> supported by the SOVD API according to 7.7 .
^a C: SOVD <code>tags</code> are used.			

6.2.3 Path item object

[Table 23](#) provides an overview of all SOVD extension properties for an OpenAPI Path Item Object.

Table 23 — Extension Property Path Item

Attribute	Type	Convention	Description
x-sovd-applicability	Table 29	C ₁ ^a	Documents for which Entity versions and variants a particular resource or even attributes of a resource are applicable.
x-sovd-cyclic-subscription-supported	boolean	C ₂ ^b	Documents if a resource supports the creation of a cyclic-subscriptions. If the identifier is missing, the SOVD server does not support cyclic-subscriptions for the resource.
x-sovd-data-category	string	M	Documents the DataCategory of a data resource exposed via the SOVD API.
x-sovd-data-groups	string[]	C ₃ ^c	Documents the groups an individual data resource belongs to. The type of the value is an array of ValueGroup .
x-sovd-fault-display-code	string	C ₄ ^d	Documents the display code of a fault. See attribute <code>display_code</code> in Fault for more details.
x-sovd-fault-severity	integer	C ₅ ^e	Documents the severity of a fault. See attribute <code>severity</code> in Fault for more details.
x-sovd-fault-scope	string	C ₆ ^f	Documents the scope for which a fault is defined. See attribute <code>scope</code> in Fault for more details.
DEPRECATED x-sovd-lock-required	boolean	C ₇ ^g	Used instead of <code>x-sovd-lock-scopes</code> to document that all lockable resources of the Entity are locked when a lock is created and no scoped locking is supported. For details on locking and its effect on child and parent entities see 7.17 .
DEPRECATED x-sovd-lock-scopes	string[]	C ₈ ^h	Contains a list of all scopes a client may acquire to lock the resource. A client shall acquire the lock for at least one of the listed scopes. For details on locking and its effect on child and parent entities see 7.17 .
x-sovd-name	string	M	Documents a user-friendly name of a resource.
DEPRECATED x-sovd-required-modes	Map of string[] (defined as <code>additionalProperties</code> see [20])	C ₉ ⁱ	Documents the list of required modes which shall be set to an Entity in order to invoke the method defined by this Path Item Object, e.g., writing an ECU configuration.
DEPRECATED x-sovd-supported-scopes	string[]	C ₁₀ ^j	Lists which lock scopes are defined for an Entity . For details on locking and its effect on child and parent entities see 7.17 . The attribute can only be defined for path items representing an Entity .
x-sovd-tags	string[]	C ₁₁ ^k	Documents the SOVD tags of a resource exposed via the SOVD API according to 7.7 .
^a C₁: the resource or Entity has at least one applicability assigned to it. ^b C₂: the resource supports cyclic subscription. ^c C₃: the resource has at least one group assigned to it. ^d C₄: a <code>display-code</code> has been assigned to a fault. ^e C₅: a <code>severity</code> has been assigned to a fault. ^f C₆: a <code>scope</code> has been assigned to a fault. ^g C₇: any action (HTTP method) on the resource would require a lock. ^h C₈: any action (HTTP method) on the Entity or resource requires a lock and defines at least one scope. ⁱ C₉: the Entity has at least one mode assigned to it. ^j C₁₀: the Entity has at least one scope assigned to it. ^k C₁₁: If the resource or Entity has at least one tag assigned to it.			

6.2.4 Operation object

[Table 24](#) provides an overview of all SOVD extension properties for an OpenAPI Operation object.

Table 24 — Extension Property Operation

Attribute	Type	Convention	Description
x-sovd-asynchronous-execution	boolean	C ₁ ^a	Documents whether the execution of the operation is asynchronous or not.
x-sovd-lock-breakable	boolean	0	true specifies that the lock on the resource can be broken by a privileged client. The default value is false which states that the lock is unbreakable.
DEPRECATED x-sovd-capabilities	Capability	M	Documents the list of SOVD capabilities a specific operation of an Entity supports at the respective methods available at the SOVD API.
x-sovd-lock-required	boolean	C ₂ ^b	Used instead of x-sovd-lock-scopes to document that all lockable resources of the Entity are locked when a lock is created and no scoped locking is supported. For details on locking and its effect on child and parent entities see 7.17 .
x-sovd-lock-scopes	string[]	C ₃ ^c	Contains a list of all scopes a client may acquire to lock the resource. A client shall acquire the lock for at least one of the listed scopes. For details on locking and its effect on child and parent entities see 7.17 .
x-sovd-proximity-proof-required	boolean	C ₄ ^d	Documents that the execution of an Entity operation requires proof of co-location.
x-sovd-required-modes	Map of string[] (defined as additionalProperties see [20])	C ₅ ^e	Documents the list of required modes which shall be set to an Entity in order to invoke the method defined by this Operation object, e.g., writing an ECU configuration.
x-sovd-tags	string[]	C ₆ ^f	Documents the SOVD tags of a resource exposed via the SOVD API according to 7.7 .
^a C₁: the operation requires an asynchronous execution. ^b C₂: any action (HTTP method) on the resource would require a lock. ^c C₃: any action (HTTP method) on the Entity or resource requires a lock and defines at least one scope. ^d C₄: the execution of the operation requires a proof of co-location. ^e C₅: the Entity has at least one mode assigned to it. ^f C₆: if the resource or Entity has at least one tag assigned to it.			

6.2.5 Schema object

[Table 25](#) provides an overview of all SOVD extension properties for an OpenAPI Schema object.

Table 25 — Extension Property Schema

Attribute	Type	Convention	Description
x-sovd-translation-id	string	C ₁ ^a	Documents a translation identifier for a property within a JSON schema, e.g., the name of a resource.
x-sovd-enum-translation-ids	Map of string[] (defined as additionalProperties see [20])	C ₂ ^b	Documents the translation identifiers for enumeration values. The key of the map shall refer to the enum value and the value shall express the corresponding translation-id.
x-sovd-unit	SovdUnitType	C ₃ ^c	Documents the unit of a property within a JSON schema.
^a C₁: an internationalized text is required for the property. ^b C₂: an internationalized text is required for the enumeration. ^c C₃: the property uses no base quantity.			

Requests and responses may contain key-value pairs which cannot be defined by the standard as they are implementation specific. An example for this is the attribute `x-sovd-enum-translation-ids` defined

in [Table 25](#). The schema then must define the keys of the map explicitly. In the SOVD JSON schema, this is expressed by using the property `additionalProperties` as defined by JSON schema^[20].

6.2.6 Header object

[Table 26](#) provides an overview of defined SOVD headers that are used across the SOVD API methods.

Table 26 — Extension Property Header

Attribute	Type	Convention	Description
Authorization	string	C ₁ ^a	The authorization header is a request header, which contains a valid token. Note: Further headers may be used as well.
Location	string:uri-reference	C ₂ ^b	The location header is a response header, which provides an absolute URI pointing to the created resource.
x-sovd-mode	Map of string[] (defined as <code>additionalProperties</code> see [20])	C ₃ ^c	The x-sovd-mode header can be used for Hint-based mode control (see 7.16.5). The key of the map shall refer to the mode-id and the value shall express the target mode to be set.
^a C ₁ : the access to the resource requires authorization. ^b C ₂ : an information about the resource location of a newly created resource is required, which allows an re-direct by the client. ^c C ₃ : hint-based mode control on the underlying Entity is supported and required.			

6.2.7 Query Parameter object

[Table 27](#) provides an overview of defined SOVD query parameters which are used across the SOVD API methods to detail resource requests or specify filter criteria for the returned responses. For serializing query parameter values, SOVD follows the default definitions of OpenAPI Specification. Therefore, query parameters shall be specified using the “form” style with “explode=true”. Based on that, query parameter values of type array or object are represented by repeating the corresponding query parameter for each value of the array or property of the object. Every element of an array is then considered as an OR condition for a query.

Table 27 — Extension Query Parameter

Attribute	Type	Convention	Description
include-schema	boolean	0	The query parameter <code>include-schema</code> allows to specify whether the response should include <code>schema</code> information in form of a <code>schema</code> property value or not. The <code>schema</code> attribute shall include the definition of all properties in the response. If the query parameter <code>include-schema</code> is set to <code>true</code> , the <code>schema</code> properties as defined by the specification is set on all objects in the response. Default: <code>false</code> . Note: A property may include its own <code>schema</code> property to be more precise on the type definition for the object in which it is contained.
tags	string	C ^a	The query parameter <code>tags</code> allows to filter available resources based on the <code>tags</code> they have specified. Tag names are case sensitive.
^a C: filtering for <code>tags</code> is required and <code>tags</code> are specified in the capability description.			

6.2.8 x-sovd-tags object

An x-sovd-tags object allows to specify a list of `tags` which can be used across the SOVD API methods to tag **Entities** and their resources as well as specify filter criteria for querying **Entity** collections and **Entity** resource collections.

To specify a list of SOVD `tags` within an SOVD capability description an x-sovd-tags object shall be added. Within the object a list of `tags` can be specified. For the tag definition itself, SOVD follows the tag object definition of the OpenAPI Specification with the defined `SOVDTag` type.

For further information about x-sovd-tags, please refer to [Annex B](#).

[Table 28](#) specifies the `SOVDTag` type.

Table 28 — SOVDTag type

Attribute	Type	Convention	Description
name	string	M	Name of the tag. This is used as an identifier to refer to the tag definition.
description	string	O	An optional description of the tag, e.g., its purpose or context.
translation_id	String	O	Identifier for translating the name.

For further information about x-sovd-tags, refer to [Annex A](#).

Example:

Definition of SOVD `tags` within a capability description

```
{
  "openapi": "3.1.0",
  "info": {},
  "tags": {},
  "servers": [],
  ...
  "x-sovd-tags": [
    {
      "name": "OBD",
      "description": "Enables tagging of OBD-related resources."
    },
    {
      "name": "ePTI",
      "description": "Enables tagging of ePTI-related resources."
    }
  ],
  "paths": {},
  ...
}
```

6.3 Online and offline capability descriptions

6.3.1 General

[Subclause 6.3](#) describes how an SOVD client learns about the diagnostic capabilities of an ExVe. Technically, the capabilities are described in an OpenAPI specification. SOVD distinguishes between two different scenarios.

In the first scenario, a diagnostic tool developer wants to develop an SOVD client for a complete series including all possible variants. Thus, the developer needs upfront information about the diagnostic capabilities which might be available on an ExVe. For this scenario, this document defines an offline capability description (see [6.3.2](#)), which contains complete information about all diagnostic capabilities and for which ExVe variants they are applicable (see [6.4](#)).

In the second scenario, a diagnostic tool developer develops a full generic diagnostic SOVD client without prior knowledge about the vehicle. For this case, this document defines an online capability description (see

[6.3.3](#)), which can be read from the ExVe by the SOVD client. The returned OpenAPI specification can be used by the SOVD client to identify the available **Entities**, resources and which methods can be applied to them.

An SOVD capability description is a well-formed and full specified OpenAPI specification. This allows SOVD clients to handle online capability descriptions and offline capability descriptions in the same way as well as use the same tooling and mechanisms for parsing and utilizing their content. The main difference is the scoped context on the specified parts of the API (see [7.5.2](#)).

6.3.2 Offline capability description

An offline capability description provides the documentation for all SOVD methods of an ExVe and therefore specifies how to access and interact with all available **Entities**, e.g., **Components** or **Apps**.

6.3.3 Online capability description

The SOVD online capability description is a mechanism which allows clients to query how to interact with a specific resource provided by the SOVD server. It is full compatible to an offline capability description, limited to a single resource.

6.4 Variant handling in offline capability description

6.4.1 General

Traditionally, vehicle projects are developed with multiple variants of ECUs with changes to the functional specification based on vehicle model, features opted by the customer, generational evolution of ECUs, etc. These variations in ECU software specification in-turn result in changes in the diagnostic specifications. The SOVD API distinguishes between two distinct aspects to resemble common automotive concepts:

- Evolution of ECU firmware: The evolution of the ECU's firmware implies that multiple versions of a firmware exist which extend or alter the diagnostic capabilities of the ECU. Different versions of a firmware can be identified using resources like "softwareVersion" and others to distinguish one version of an ECU's firmware from another. Throughout this section this aspect is referred to as **Entity** versions.
- Optional features: Diagnostic capabilities of an ECU can depend on the existence of certain features in the vehicle, e.g., the value of the configuration resource "accSpeedLimit" for the speed limit of an adaptive cruise control depends on the sensors built into the vehicle. In the following, this is referred to as **Entity** variants.

There might be other factors influencing which aspects are considered to specify variants and/or versions depending on the chosen kind of variant handling.

The major difference between **Entity** versions and variants is that resources depending on a particular **Entity** version are absent if the **Entity** version does not match. However, resources depending on an **Entity** variant, may exist if the **Entity** variant does not match, but do not return reasonable values or an error.

In SOVD, two types of **Entities** – **Components** and **Apps** – provide the information which **Entity** version and variant is built into the extended vehicle. The extended vehicle's capability description provides the information which resource is applicable for which **Entity** version or variant using an OpenAPI custom extension (see [6.4.2](#)).

Note In general, the process to identify and manage **Entity** variants is proprietary and depends heavily on tools, use cases, engineering processes, rules, etc. of the ExVe manufacturer. The mechanisms described in this section may not cover all scenarios.

6.4.2 Extension: x-sovd-applicability

SOVD introduces the custom extension `x-sovd-applicability` that can be defined in the offline capability description for defining for which **Entity** versions and variants a particular resource or even attributes of a request or response are applicable. The extension attribute enumerates **Entity** versions and variants which support a specific resource (Path object). For ExVe manufacturers with the strategy to use one capability

description per vehicle variant, `x-sovd-applicability` would be defined as part of the Open API Info object. [Table 29](#) defines the attribute `x-sovd-applicability` in detail.

Table 29 — `x-sovd-applicability` definition

Attribute	Type	Convention	Description
versions	SourceValueMatch[]	Ca	Version identifiers for which the parent element (i.e., path, method, ...) is applicable. If more than one version identifier is given, all of them shall match.
variants	SourceValueMatch[]	Ca	Variant identifiers for which the parent element is applicable. If more than one variant is given, all of them shall match.

^a C: at least one of the attributes shall be set. Otherwise, the attribute `x-sovd-applicability` would be absent, and the parent element exists for all versions or variants. If both attributes are set, both shall match.

[Table 30](#) specifies the `SourceValueMatch` definition.

Table 30 — `SourceValueMatch` definition

Attribute	Type	Convention	Description
source	string:uri-reference	M	Reference to the source from which the information can be obtained. The URI reference may reference attributes of a resource or an Entity using the notion <code><URI reference to source>#<JSON pointer></code> to reference any value returned. The GET method shall be supported by the referenced source.
match	MatchType	0	Enumeration defining how the value returned by the source shall be matched against the value given in the attribute values. If not set, the attribute value defaults to <code>equals</code> .
values	AnyValue[]	M	The value returned by the source is matched against the values in the array based on the specified match type. At least one of the values shall match.

[Table 31](#) specifies the `MatchType` enumeration.

Table 31 — `MatchType` enumeration

Value	Description
<code>equals</code>	The value returned by the referenced source shall at least match one element of the attribute values.
<code>not_equals</code>	The value returned by the referenced source shall not match any value in the attribute values.
<code>regexp</code>	Each element of the attribute values defines a regular expression (see ECMA 262, [14]) and the value returned by the referenced source shall match at least one of the regular expression. ^a
<code>semver</code>	The value returned by the referenced source shall match the contents of the attribute values by comparing it against a semantic version comparator as defined by Node.JS NPM [29] like <code>1.2.3 - 2.3.4</code> or <code>1.2.x</code> . ^a
<code>rule</code>	The value shall be matched using an ExVe manufacturer specific rule like <code>A34 + B12</code> .

^a In this case, the values attribute typically contains a single element.

6.4.3 Usage of `x-sovd-applicability` by the SOVD client

A client shall use the information in the offline capability description to define the behavior of the application based on the versions and variants mounted in the vehicle. The variant information as well as the version information can be obtained from the referenced sources. Once the version and variant information is known, the client can use this information to determine from the `x-sovd-applicability` extension in

the capability description the resources matching the vehicle to be diagnosed. The client is responsible for fetching the values from the sources defined by `x-sovd-applicability` as well as for performing the comparison defined by the `match` attribute.

The following snippets of a capability description for the resources `/components/PowerSteering/data/SteeringMonitors` and `/components/PowerSteering/data/SteeringWheelRotationSpeed` serves as an example what can be accessed by the client:

```
/components/PowerSteering/data/SteeringWheelRotationSpeed:
  x-sovd-applicability:
    variants:
      source: /components/PowerSteering#variant
      values: ["Pow_Str_Variant_High"]
    versions:
      source: /PowerSteering/data/swVersion
      match: semver
      values: ["1.0.x"]

/components/PowerSteering/data/SteeringMonitors:
  get:
    responses:
      "200":
        content:
          application/json:
            schema:
              type: object
              properties:
                SteeringTorque:
                  type: float
                SteeringWheelAngle:
                  type: float
                SteeringWheelRotationSpeed:
                  type: float
                  x-sovd-applicability:
                    versions:
                      source: components/PowerSteering/data/swVersion
                      match: semver
                      values: [ "1.1.x" ]
```

The resource `SteeringWheelRotationSpeed` is only available in case the variant `Pow_Str_Variant_High` is available and the resource `swVersion` is any of `1.0.x`. Whether this variant is available or not can be determined by executing the method `GET /components/PowerSteering` and evaluating the response attribute `variant`. In a later version of the software (`swVersion 1.1.x`) the `SteeringWheelRotationSpeed` is contained within the resource `SteeringMonitors`.

7 SOVD REST API

7.1 General

[Clause 7](#) provides an introduction to the standardized SOVD API methods. For each API method, the underlying path and query parameters, request, and response headers as well as request and response bodies and the resulting response status codes of the API methods are specified. Standardized HTTP request and response headers are only specified if they are explicitly required from an SOVD viewpoint, like for instance authorization. In addition, request headers are defined as required on top of standard HTTP headers.

In case an SOVD server API requires optional request parameters, the API shall guarantee the response, even if optional parameters are not set by SOVD Client.

7.2 Definition of conventions for API methods

7.2.1 Query parameters

The convention optional means that the parameter may be set by the SOVD client. In case the parameter is not set, the default is used by the SOVD server, if described. The evaluation of the parameter is mandatory for the SOVD server.

Different query parameters are defined within the API methods, which are used as a filter to narrow down the result of an API method call. This document defines the syntax for the combination of query parameters as follows:

- If different query parameters are concatenated using "&" this means that a logical "AND" is applied. I.e., all the query parameters shall be fulfilled.
- If the same query parameter is concatenated using "&", a logical "OR" is applied. I.e., at least one of the query parameters shall be fulfilled. This applies to arrays and properties of objects.

This definition is independent from the order of the query parameters. The logical OR is closer binding than logical AND.

Deviations from the behavior specified above are described at the respective methods.

7.2.2 Request body

The convention optional means, that the parameter is supported by the SOVD server, when defined in the capability description. In this case the SOVD client shall set the parameter.

7.3 Timeout definitions

The SOVD API definition does not define timeouts. However, there are topics which are not covered by the regular HTTP protocol timeouts and thus these timeouts are implementation specific. An SOVD server implementation shall provide the timeout in seconds then as part of the response which initiated the action.

The `x-sovd-retention-timeout` extension is defined as follows: The time after which the temporary resource is no longer available to an SOVD client. The starting point of the timer depends on the API method called and is defined in [Table 32](#).

Note The SOVD API addresses the ExVe perimeter and not only the physical vehicle.

Table 32 — Timeout definition header

API method	Retention timer starts when
POST /{entity-path}/operations/{operation-id}/executions	Status of operation changed to <code>completed</code> or <code>failed</code> .
POST /{entity-path}/cyclic-subscriptions	Cyclic data reception terminated, e.g., because the subscription expired.
POST /{entity-path}/triggers	Trigger-based access terminated, e.g., because the lifetime expired.
POST /{entity-path}/scripts/{script-id}/executions	Status of the script execution changed to <code>completed</code> , <code>failed</code> or <code>terminated</code> .
PUT /updates/{update-package-id}/{prepare execute}	Status of the update package changed to <code>completed</code> or <code>failed</code> . Once the execute step has been requested, the retention time is reset.
PUT /updates/{update-package-id}/{automated}	Status of the update package changed to <code>completed</code> or <code>failed</code> .
POST /updates	Update package resource created in SOVD server.

7.4 API methods for access of supported SOVD server versions

7.4.1 Read all supported versions of the SOVD server

Method:

GET `https://{sovd-server-host}/{exve-manufacturer-specific}/version-info`

Method Description:

This method provides information about all supported versions of the SOVD standard.

Path Parameters:

[Table 33](#) specifies the path parameters to read all supported versions of the SOVD server.

Table 33 — Path parameters — Read all supported versions of the SOVD server

Parameter Name	Type	Convention	Description
sovd-server-host	string	M	Host name of the SOVD Server as per RFC 6763 ([19]).
exve-manufacturer-specific	string	O	Optional path element to avoid clashes in the URIs provided by in-vehicle HTTP servers.

Query Parameters:

[Table 34](#) specifies the query parameters to read all supported versions of the SOVD server.

Table 34 — Query parameters — Read all supported versions of the SOVD server

Parameter Name	Type	Convention	Description
include-schema	boolean	O	Specifies whether the response should include schema information or not. The default is <code>false</code> .

Request Headers:

This method does not use specific header parameters.

Response Status Codes:

[Table 35](#) specifies the response status codes parameters to read all supported versions of the SOVD server.

Table 35 — Response status codes — Read all supported versions of the SOVD server

Status Code	Response Body	Description
200	Table 36	The request was successful.

Response Body:

[Table 36](#) specifies the response body parameters to read all supported versions of the SOVD server.

Table 36 — Response body — Read all supported versions of the SOVD server

Attribute	Type	Convention	Description
sovd_info	SOVDInfo[]	M	SOVD specific information
schema	OpenAPI Schema Object	M	Schema definition of the response

[Table 37](#) specifies the SOVDInfo type.

Table 37 — SOVDInfo type

Attribute	Type	Convention	Description
version	string	M	Version of the SOVD standard supported by the SOVD server as a semantic version string (see [44])
base_uri	string:uri-reference	M	Version specific base URI for interacting with the SOVD server.
vendor_info	VendorInfo	0	Vendor specific information

[Table 38](#) specifies the `VendorInfo` type.

Table 38 — VendorInfo type

Attribute	Type	Convention	Description
version	string	M	Version of the SOVD server (vendor specific string).
name	string	M	Name of the vendor of the SOVD server.

The `VendorInfo` type can include additional vendor specific information describing the SOVD server implementation in more detail.

7.5 API methods for access to capability description content

7.5.1 General

The purpose of an *online capability description* is to get all required information to use a specific SOVD method, in a self-contained manner and without requiring an overall offline capability description. Therefore, the SOVD API allows to request such *online capability descriptions* at individual entities and resources of the SOVD API to provide corresponding online documentation to SOVD clients. For example, if an *online capability description* is requested at a specific operation resource of an **Entity**, the SOVD API returns an OpenAPI specification, which describes all relevant information needed for interacting with this operation resource. This comprises, for example,

- the supported HTTP methods of the operation
- related request and response structures
- defined HTTP response status codes

as well as SOVD-specific extensions, e.g., the supported capabilities of the operation or `translation_ids` for parameters within requests or responses.

7.5.2 Retrieve online capability description

Method:

`GET /{Any path}/docs`

Method Description:

For **Entities**, resources, and resource collections, a corresponding online capability description can be requested at the SOVD API. The resulting online capability description is a self-contained, valid OpenAPI specification. The description contains information which refers to the creation, reading, updating, or deleting of the respective element as defined in [6.3.3](#).

An online capability description for a data resource with url (`/ {entity-path} /data/ {data-id}`) can be requested using path `/ {entity-path} /data/ {data-id} /docs`. Moreover, for each supported HTTP method all defined HTTP response status codes are documented together with the data model of the resulting response payloads.