ASAM SOVD

Service-Oriented Vehicle Diagnostics

API Specification

Version 1.0.0

Date: 2022-06-30

Base Standard

Table of Contents

Foreword

The SOVD Standard provides an API for diagnosing software-based vehicles. It provides uniform access to the diagnostic content of HPCs and their related applications as well as classical ECUs.

SOVD follows an HTTP REST based approach. Thereby no automotive specific stack is needed on client side. Due to the flexible type-system used, it provides access to a broad variety of content required for HPC diagnostics.

SOVD supports the following scope:

- Capability discovery
- Reading and deletion of fault entries
- Reading and writing of data resources
- Reading and writing of configurations
- Control of operations (including control of entity states via defined modes and locking of entities)
- Software update
- Handling of bulk data
- Logging data access

For the convenience of the user, a machine-readable OpenAPI definition of the methods is published alongside this document.

1 Introduction

1.1 Overview

The ASAM SOVD standard defines an API which standardizes the methods for diagnosing HPCs and classical ECUs, the retrieval of the diagnostic capabilities, and the discovery of the SOVD methods in a vehicle. The SOVD API provides a unified access to classic ECUs and HPCs. This access can be performed remotely (e.g., backend or cloud), in the workshop (e.g., workshop tester), or in the vehicle (e.g., onboard tester).

The SOVD API leverages existing technologies as described in chapter 2.1.

The SOVD API provides the following functions:

- Clients can access the faults, including reading the fault entries, reading environment data, and deleting fault entries.
- Measurements and identifications from all entities in the vehicle can be read. In addition, identifications may be written as well.
- SOVD supports the execution of routines, I/O controls, and software functions. Their execution can only be performed in certain modes or states. Thus, an SOVD client can set the Component into a specific mode.
- The configuration of a vehicle (e.g., equipment, country, customer demand, variant coding etc.) can be read and written using the SOVD API.
- SOVD encapsulates the OEM specific software update strategy (incl. FOTA) through a generic API.
- Access to logging information of an HPC.

With these features SOVD can cover all areas of the vehicle life cycle: Engineering (Development), Manufacturing (Production), After Sales (Maintenance and Repair), and Vehicle operation (Use).

There are several aspects which are not covered by the SOVD standard, 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
- Load balancing of concurrent requests

1.2 Motivation

Vehicle electronics topologies are changing rapidly. New functions such as autonomous driving require application-level software of high complexity to be executed within the vehicle.

The introduction of HPCs in vehicles is associated with changes in the classical E/E vehicle architectures. While classical distributed embedded ECU architectures were used previously, domain- or zone-based E/E architectures dominate today.

On HPCs, there will be an agile exchange of applications to enable new functionalities for the user. Since the HPCs can be compared to classic computers running numerous applications,

this means that diagnostics are no longer limited to classic fault codes (DTCs with environment data). An analysis of the running programs is required in real operation. The diagnostic of these programs is comparable to the error and runtime analysis of software running on PCs. That means, it has the memory usage, processor load and number of active services that is required to be recorded, and therefore also log and trace files might be needed.

This shifts the focus from ECU diagnostics in today's vehicles from checking hardware to checking the software functionality of applications, which corresponds to a paradigm shift.

All these topics result in new challenges regarding the management of the vehicle life cycle, which includes the diagnostic, (re-)configuration and re-programming of vehicle electronics.

New operating systems also no longer follow the classic rules for embedded systems, therefore, the diagnostic changes and adapts due to several factors, these include:

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

The SOVD standard takes these points into account and provides a diagnostic API based on current IT technologies for the purpose of diagnosing vehicles with modern E/E architectures.

2 Relations to Other Standards

2.1 References to Other Standards

The SOVD API uses the following standards and technologies:

- The API follows the REST principles [16] and uses JSON [28] for encoding the transmitted data.
- SOVD is designed to work with HTTP/1.1 [23] but for achieving the best communication performance HTTP/2 [22] is recommended. No HTTP/2 specific features are used.
- The SOVD API utilizes the OpenAPI [10] specification to define the API as well as the diagnostic capabilities of the vehicle.
- The authentication and authorization of clients builds upon OpenID Connect and OAuth 2.0 [12], [13], [14], but an OEM may use other authentication mechanism like certificates if required.
- ASAM data types [35] are mapped to JSON types.

3 SOVD Architecture

3.1 Usage Scenarios

As wireless connectivity technologies enter the automotive marketplace there will be new methods of accessing and using vehicle diagnostics. Historically, vehicle diagnostics have been focused on proximity-based use cases, where a technician or worker is located close to the vehicle and connects an external device to perform diagnostic tasks. Wi-Fi and mobile broadband network access (4G, 5G) allow for remote and OTA use cases such as remote diagnostics, configuration, and software updates. With the introduction of HPCs into new vehicle architectures this also gives the ability to perform in-vehicle diagnostic tasks without any permanent external connectivity.

SOVD offers solutions for in-vehicle, proximity and remote diagnostics use cases. This standard intends to meet the requirements for those use cases, which are not covered by current diagnostic stacks, based on UDS (ISO 14229 [1]), D-PDU API (ISO 22900-2 [18]), ODX (also known as ISO 22901-1 [9] or ASAM MCD-2D) and ASAM MCD-3D (also known as ISO 22900-3 [29]).

3.1.1 Remote

For remote use cases it is assumed that the operator is not close to the vehicle but accesses the vehicle remotely via (mobile) broadband network (over-the-air). Remote use cases do not necessarily require a human end-user but can also include cloud-based services such as fleet management, data collection and analysis that are only indirectly feeding information to a human user. Remote use cases may rely on onboard use cases, e.g., to buffer data in cases where the network is not available.

Remote diagnostics use cases include but are not limited to the following:

- Information retrieval (e.g., fuel level, battery health check)
- Checking operational status
- Configuration
- Emissions check
- Workshop repair preparation
- Remote troubleshooting
- Remote activation of vehicle functions
- Software update
- Firmware update
- Fleet management
- Configuration of the software

3.1.2 Proximity

For proximity use cases it is assumed a technician or worker is close to the vehicle and has a test device connected to a particular vehicle (by cable or wireless).

Proximity diagnostics use cases include but are not limited to the following:

- Troubleshooting
- Performing actuations or stimulations for functional checks
- Periodical technical inspection

- Emissions check
- Checking operational status
- Software update
- Firmware update
- Configuration of the software

3.1.3 In-Vehicle

For in-vehicle use cases it is assumed that these use cases can run autonomously within the vehicle without a permanent connection to a remote server or a proximity tester. However, results of onboard use cases may also be accessed by proximity or remote use cases.

In-vehicle diagnostics use cases include but are not limited to the following:

- In-vehicle monitors
- Preventive/Predictive Maintenance
- Fleet monitoring scenarios (e.g., periodically collecting vehicle status)

4 SOVD API General Aspects

4.1 HTTP REST Based Approach

The SOVD API follows the REST principles [16]. This means that diagnostic content is provided in 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 4.2) and the standardized resources and resource collections provided for that entity (see 4.3).

The operations available for the individual diagnostic content are represented using HTTP methods. SOVD makes use of the following HTTP methods:

Table 1 HTTP methods used by SOVD

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

The following example shall briefly illustrate the REST based approach used by SOVD. Details on how the resource structure is built are given in the following chapters.
The example shows a software application in the vehicle providing functionality to control the windows, for example it offers information about the “RearWindows” positions.

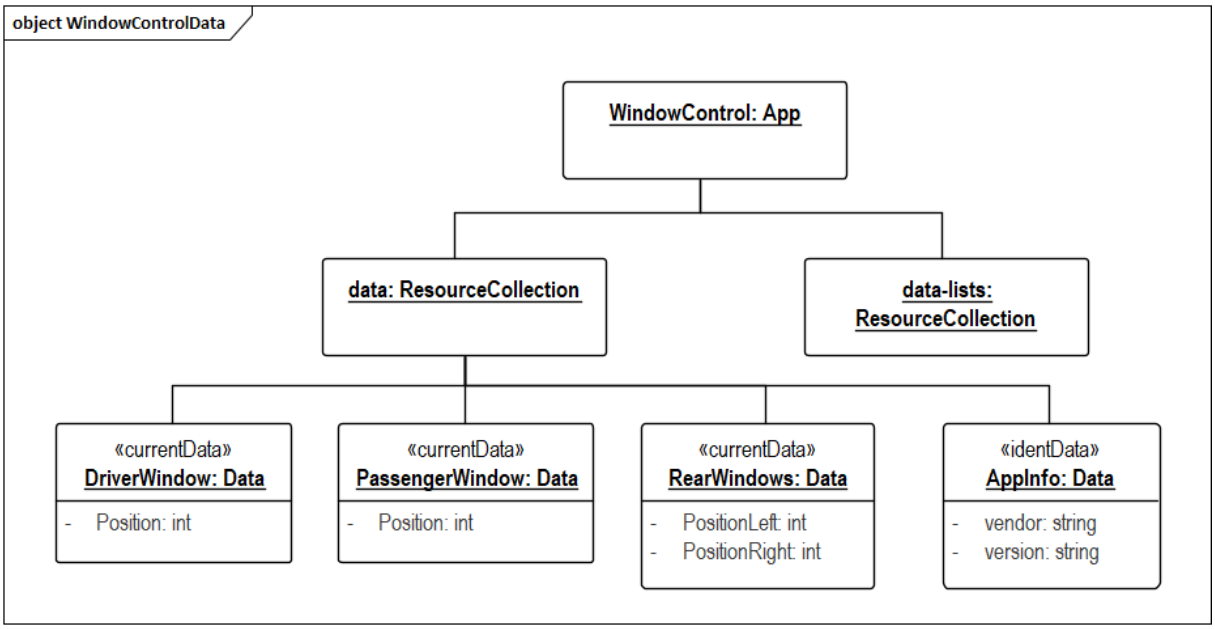

Figure 1 WindowControl example

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 will be explained in the upcoming chapters 4.2 and 4.3.

Example:

Request:

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

4.2 Entities

The SOVD API defines entities holding diagnostic content. To access diagnostic content, an SOVD client needs to specify the entity it wants to interact with. This subchapter covers the entity types, the hierarchy, and how to reference the entity in a method request.

4.2.1 Entity Types

The SOVD standard defines the entity types as shown in Figure 2. The entity types are defined flexibly enough to support various topologies and diagnostic strategies. SOVD entities hold the diagnostic content as resources. Resource collections available for an entity type are defined in 4.3.1.

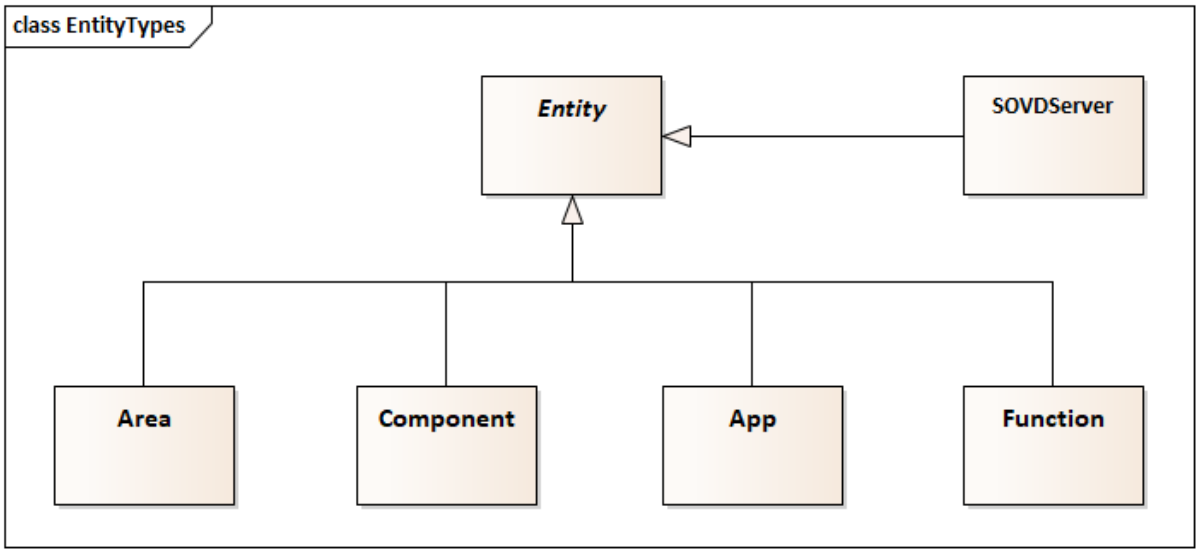

Figure 2 Entity types overview

Table 2 describes the different entity types in more detail.

Table 2 Details of entity types

EntityType	EntityCollectionType	Description
SOVDServer		An ***SOVDServer*** represents the entry point of the API structure. Resources on that level are only related to the SOVD server itself and not aggregated from other entities.
Area	`areas`	An ***Area*** represents a logical view and can be used to describe various vehicle topologies, such as domain architectures (body control, Infotainment, powertrain), zone architectures (front, rear), or logical architectures (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). ***Component***s representing hardware are characterized by the availability of a CPU and execute software. Examples are HPCs, ECUs, and Smart sensors. Software as part of ***Component***s is characterized as basic software required to execute ***App***s. Examples for basic software are operating systems, or Hypervisor. Only software ***Component***s or ***Component***s containing software parts shall link to ***App***s.
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 OEM to define access to diagnostic information, which may be spread across several ***Component***s, e.g., Vehicle Identification. **Note:** This standard defines no further details for the handling of ***Function***s.

From an API perspective, there is no differentiation between hardware and software ***Component***s. However, an OEM may choose to do so and represent this within a component hierarchy.

4.2.2 Multi-SOVD Server Environments

This standard neither defines nor demand a specific architecture within a vehicle. It is intended to support any architecture defined by an OEM, including multiple SOVD servers in a single vehicle.

It is strongly recommended, that each entity in a vehicle is bound to and can only be accessed through a dedicated SOVD server (see 4.2.1).

4.2.2.1 Public SOVD Server

Each SOVD server can be directly accessible from the outside through connectivity measures (remote or proximity). It is within the responsibility of the OEM to provide identification of a Public SOVD server for remote SOVD clients. For proximity clients, the identification method as described in 4.9 shall be used.

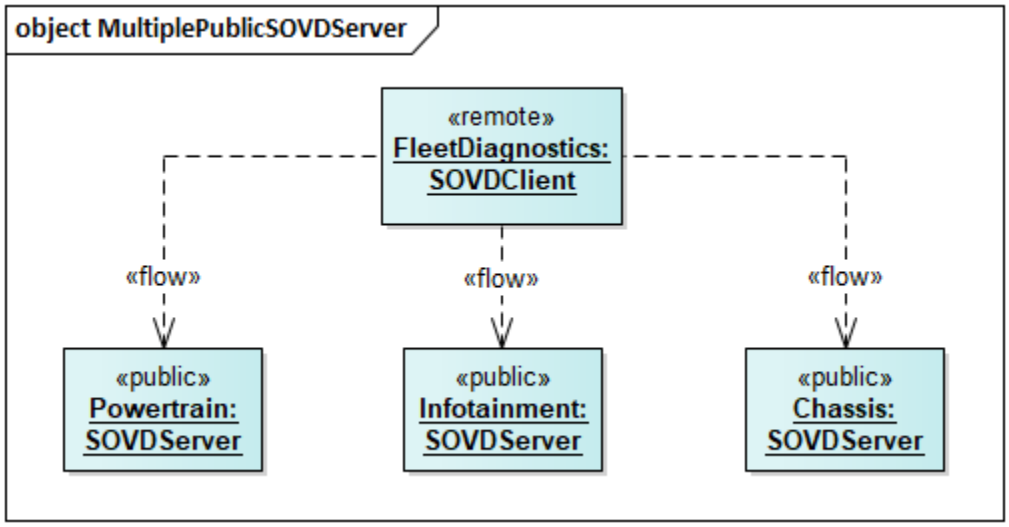

Figure 3 Public SOVD server

4.2.2.2 Private SOVD Server

An SOVD server can also be restricted to be accessible from the in-vehicle network. Direct access to a Private SOVD server is then only possible for an On-Board SOVD client. Diagnostic Services as described in this document, which are provided by a Private SOVD server have to be forwarded by a Public SOVD server to an external SOVD client, in case these Diagnostic Services should be externally available.

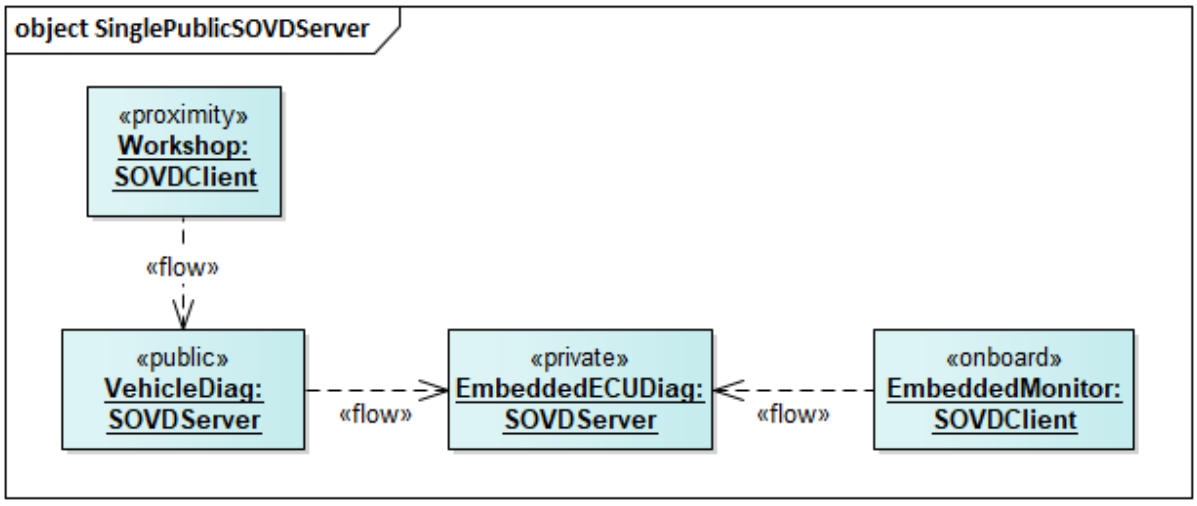

Figure 4 Private SOVD server

4.2.3 Entity Hierarchy and Relations

The entity types described above can be used to build a hierarchy for the SOVD API. Figure 5 depicts the standardized entity hierarchy. The entry point into this structure is the SOVD server. OEMs are free to select the needed entity types. For simplicity of this figure, the {`Entity-collection`}s are not shown as separate classes but can be seen from the name of the respective association ends.

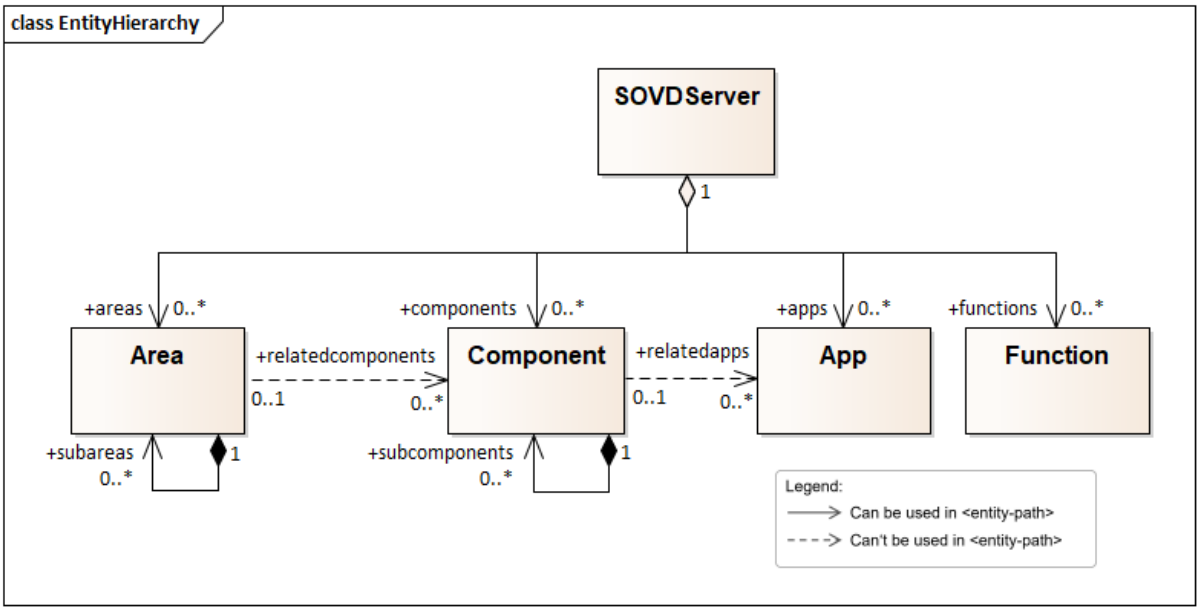

Figure 5 Entity hierarchy

An ***Area*** can have multiple subareas, which themselves are ***Area***s and thus can have subareas. ***Area***s on all levels can have links to ***Component***s. This information can be used by an SOVD client to present diagnostic information according to the topology.

A ***Component*** can have multiple subcomponents, which themselves are ***Component***s and thus can have subcomponents. ***Component***s on all levels can have links to ***App***s. This information can be used by an SOVD client to present all ***App***s running on a ***Component***.

4.2.4 Entity Identifier and Path

The standard only defines the names of the entity collection resources and the entity types. The entity identifiers and names are specified by the vehicle manufacturer when the API is realized.

Every entity is accessible via its `entity-path`. The entity path is built from the element's entity collection name 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. For resource on entity level ***SOVDServer*** the entity collection name is omitted.

Example:
`{base_uri}/components/DrivingComputer/subcomponents/Linux`

4.3 Resources

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

An OEM may specify additional resource types, collections, and top-level-resources for an entity. To avoid naming conflicts with resources, collections thereof, and resource types standardized in a later version of the SOVD API, the OEM shall use the prefix "`x-<oem>-`".

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.

4.3.1 Resource Collections

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

Table 3 shows 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 3 Standardized resource collections

Resource Collection	Description
configurations	Collection of configuration resources
bulk-data	Collection of bulk data resources for the respective BulkDataCategory
data	Collection of static and dynamic data resources
data-lists	Collection of combined data resources of an entity
faults	Collection of fault resources of an entity
operations	Collection of operation resources
updates	Collection of update package resources
modes	Collection of mode resources
locks	Collection of lock resources

Table 4 gives an overview on which resource collections are allowed on which entity type.

Table 4 Relation between entities and resource collections

Entity level	configurations	bulk-data	data	data-lists	faults	operations	updates	modes	locks
SOVDServer	X	X	X	X	X	X	X	X	X
Area	not supported by this standard								
Component	X	X	X	X	X	X	[1]	X	X
App	X	X	X	X	X	X	[1]	X	X
Function	not supported by this standard								

4.3.2 Standardized Resources

The resource names shown in Table 5 are defined by this standard for an entity.

Table 5 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	***App***s hosted on a ***Component***
related-components	***Component***s 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
token	Request token from the vehicle for authorization

Table 6 gives an overview, which resource is allowed to be used on which entity type.

[1] **Not allowed**

Table 6 Relation between entities and resources

Entity level	docs	version-info	logs	related-apps	related-components	data-categories	data-groups	authorize	token
SOVDServer	X	X	X	[2]	[2]	X	X	X	X
Area	X	[1]	[2]	[2]	X	[2]	[2]	[2]	[2]
Component	X	[1]	X	X	[2]	X	X	[2]	[2]
App	X	[1]	X	[2]	[2]	X	X	[2]	[2]
Function	X	[1]	[2]	[2]	[2]	[2]	[2]	[2]	[2]

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 Appendix: A.

4.4 SOVD API Versioning

The SOVD API uses URI based versioning. Thus, the version of the SOVD API is included in the URI.

For this standard version base_uri is defined as **`https://<SOVD-Server-Host>/`**<OEM specific>**`/v1/...`**. The optional path element <OEM specific> can be used to avoid clashes in the URIs provided by in-vehicle HTTP servers.

The SOVD API provides the URI **`https://<SOVD-Server-Host>/`**<OEM specific>**`/version-info`** to allow SOVD clients to determine which versions are supported by an SOVD server. The path to the version-info resource shall remain the same for this and all future versions of the SOVD API.

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 as long as this URI provides syntactically the same information.

Table 7 shows the schema of the `version-info` resource.

[2] **not supported by this standard**

Table 7 `VersionInfo` type

Attribute	Type	Convention	Description
`sovd_info`	SOVDInfo`[]`	M	SOVD specific information
`schema`	`OpenAPI Schema`	M	Schema of the version-info resource

Table 8 `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 https://semver.org)
`base_uri`	`string:uri-reference`	M	Version specific base URI for interacting with the SOVD server.
`vendor_info`	VendorInfo	O	Vendor specific information

Table 9 `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 may include additional vendor specific information describing the SOVD server implementation in more detail.

4.5 Internationalization and Localization

4.5.1 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 (as in Table 42), the translation identifier is built up by using the name of the internationalizable attribute and appending `_translation_id`. For example, in Table 42 the attribute `symptom` can be internationalized and thus the internationalization identifier is provided by the attribute `symptom_translation_id`.

4.5.2 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.

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

Table 10 `x-sovd-unit` attributes

Attribute	Type	Convention	Description
`display_name`	`string`	M	The string for displaying the unit in an SOVD client (e.g., “km/h” or “min^{-1}”.
`reference`	`string`	O	Identifier of the unit for localization, e.g., as provided by the ODX_RS_UNIT_LIB.odx-d [26].
`factor_si_to_unit`	`number`	M	Factor to calculate the value in the defined unit.
`offset_si_to_unit`	`number`	O	Offset to add to the value.
`physical_dimension`	PhysicalDimension	C	Definition of the physical dimension based on SI units Condition: Unit has physical dimension

Table 11 `PhysicalDimension` type

Attribute	Type	Convention	Description
`length`	`integer`	C_1	Exponent for length
`mass`	`integer`	C_1	Exponent for mass
`time`	`integer`	C_1	Exponent for time
`current`	`integer`	C_1	Exponent for current
`temperature`	`integer`	C_1	Exponent for temperature
`molar_amount`	`integer`	C_1	Exponent for molar-amount
`luminous_intensity`	`integer`	C_1	Exponent for luminous intensity

- C_1: Only set if dimension is not 0.

Example:

```
x-sovd-unit:
  display_name: 'km/h'
  factor_si_to_unit: 3.6
  offset_si_to_unit: 0
  physical_dimension:
    length: 1
    time: -1
```

4.5.3 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 [19] 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 any localization.

4.6 Response Status Codes

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 successfull 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.

4.6.1 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 12: 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 13: 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.
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 14: 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.
504	Gateway Timeout	No response received from the underlying entity (e.g., no UDS response from an ECU) in a given amount of time.

4.6.2 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 chapter 6 use the GenericError defined here to provide this additional information.

Table 15 `GenericError` type

Attribute	Type	Convention	Description
error_code	`string` (see ErrorCode)	M	SOVD standardized error code
vendor_code	`string`	C	Vendor specific error code Condition: If the value of `error_code` is `vendor-specific` this attribute has to be filled.
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.

On some occasions an error applies only to parts of a response, e.g., when reading a data resource. In this case the type DataError is used which includes a path to the erroneous attribute of the response:

Table 16 `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	Describes the error more precisely. Condition: The attribute referenced by path contains an error.

4.6.3 SOVD Error Codes for Common Cases

Table 17 SOVD error codes for common cases

ErrorCode	Description
vendor-specific	Details are specified in the vendor_code
not-responding	The ***Component*** which handles the request (e.g., an ECU) has been queried by the SOVD server but did not respond.
error-response	The ***Component*** receiving the request has answered with an error. For UDS, the message should include the service identifier (Key: 'service' and Value of type `number`) and the negative response code (Key: 'nrc' and Value of type `number`).
invalid-signature	The signature of the data in the payload is invalid.
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.
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.
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 a regular operation of the SOVD server cannot be expected.
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.
insufficient-access-rights	The SOVD client does not have the right to access the resource.
precondition-not-fulfilled	The preconditions to execute the method are not fulfilled.
update-process-in-progress	An update is already in progress and not yet done or aborted.
update-automated-not-supported	Automatic installation of update is not supported.
update-preparation-in-progress	An update is already in preparation and not yet done or aborted.
update-execution-in-progress	Another update is currently executed and not yet done or aborted.

4.7 Common Primitive Data Types

Table 18 presents the list of common JSON data types used within the SOVD API in addition to those defined in the OpenAPI specification (see [10] which itself is defined on top of the JSON value types introduced in RFC 8259 (see [28])).

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 18 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 [30])
`string`	`uri`	An absolute URI according to RFC 3986 (see [31]), e.g., {base_uri}`/components/ecu-1`
`string`	`uri-reference`	A URI reference according to RFC 3986 (see [31]), e.g., `/components/ecu-1`
`string`	`uri-template`	An URI template string according to RFC 6570 (see [32]), 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 [33]), e.g., #/data/batteryVoltage
string	uuid	A Universally Unique Identifier (UUID) according to RFC 4122 (see [34])
string	date	A full date according to RFC 3339 (see [19])
string	date-time	A date-time according to RFC 3339 (see [19])
string	time	A time according to RFC 3339 (see [19])
string	duration	A duration according to RFC 3339 (see [19])
string	period	A period according to RFC 3339 (see [19])
string	regex	Regular expressions as defined in ECMA 262 (see [17])
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.

Table 19 Mapping of ASAM data types

ASAM Data Type	JSON Type	JSON Format	Comment
A_ASCIISTRING	string	-	
A_BITFIELD	string	bit-field	The bitfield is represented as a string of 0 and 1s and not as a number.
A_BOOLEAN	boolean	-	
A_BYTEFIELD	string	byte	
A_FLOAT32	number	float	
A_FLOAT64	number	double	
A_(U)INT8 / 16 / 32	integer	int32	
A_(U)INT64	integer	int64	
A_UNICODE2STRING	string	-	

4.8 Definition of JSON Schema and Objects for Vehicle Responses

The SOVD API returns any 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 the following:

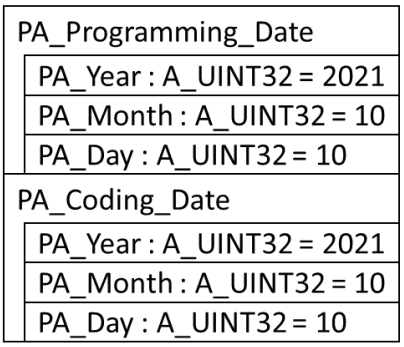

Figure 6 Example response structure

For this response structure the following example JSON schema and object is generated:

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.

4.9 Discovery of the SOVD Server for the Proximity Use Case

4.9.1 General Concept

Use Multicast DNS (mDNS, RFC6762 [20]) to obtain the IP address and DNS Service Discovery (DNS-SD, RFC 6763 [24]) 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 will not appear to be valid for a longer period.

4.9.2 Network Connection

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.

4.9.3 Address an SOVD Server Using mDNS

The public SOVD server shall implement mDNS host according to RFC6762 ([20]). 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 has to be assigned a unique hostname (e.g., by extending the VIN with (1), (2) and so on).

Note: 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: 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.).

4.9.4 Discovery of the SOVD Server Using DNS-SD

The public SOVD server shall implement DNS based service discovery as described in RFC6763 ([24]) with the following parameters:

- Protocol description: ASAM SOVD
- Primary Transport Protocol: TCP
- TXT Record key/value pairs
  - identification=ABC122345789
  - accessurl=https://ABC122345789.local:9999/vehicle

4.9.5 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.:9999 (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
```

4.10 Discovery of the SOVD Server for the Remote Use Case

In case the SOVD API is also provided for the remote use case, there are several ways how to discover and how to address individual vehicles. This part is not specified in the SOVD standard.

5 Capability Description

5.1 Introduction

This chapter introduces the structure and definitions underlying the SOVD capability descriptions. The documentation of an SOVD API exposed by an SOVD Server, in form of an SOVD capability description, builds on top of the OpenAPI Specification [10] (with version >= 3.1.0). The following subchapters will provide an overview of SOVD-specific definitions as well as conventions and recommendations for documenting an SOVD API.

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.

5.2 Capability Description Structure

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	Description
servers	Documentation of the single SOVD Server the SOVD capability description is defined for.
paths	Documentation of SOVD resources implemented by the SOVD server.
components	Documentation of the SOVD-specific definitions as well as OEM-specific reusable definitions, e.g., parameters, responses, schemas, etc.
security	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 subchapters 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 subchapter 5.2.1.

5.2.1 Info Object

Table 22 provides an overview of all SOVD extension properties for an OpenAPI `Info` object.

Table 22 Extension Property Info

Extension Identifier	Usage in SOVD	Description
x-sovd-version	Mandatory	Documents the SOVD standard version underling to the capability description.

5.2.2 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

Extension Identifier	Usage in SOVD	Description
x-sovd-lock-required	Conditional	Documents if a lock is required for the interaction with a particular resource. If specified as part of a Path Item Object, any action (HTTP method) on the resource would require a lock.
x-sovd-required-modes	Conditional	Documents the list of required modes which have to be set to an entity in order to invoke the method defined by this Path Item Object, e.g., writing an ECU configuration.
x-sovd-data-category	Conditional	Documents the category of a data resource exposed via the SOVD API.
x-sovd-name	Mandatory	Documents the name of a resource.
x-sovd-data-groups	Conditional	Documents the groups an individual data resource belongs to. The type of the value is an array of ValueGroup.
x-sovd-fault-scope	Conditional	Documents the scope for which a fault is defined. See attribute `scope` in Fault for more details.
x-sovd-fault-display-code	Conditional	Documents the display code of a fault. See attribute `display_code` in Fault for more details.
x-sovd-fault-severity	Conditional	Documents the severity of a fault. See attribute `severity` in Fault for more details.

5.2.3 Operation Object

Table 24 provides an overview of all SOVD extension properties for an OpenAPI `Operation` object.

Table 24 Extension Property Operation

Extension Identifier	Usage in SOVD	Description
x-sovd-lock-required	Conditional	Documents if a lock is required for the interaction with a particular resource. If specified as part of an Operation object, the lock is required only for performing this particular action (HTTP method, e.g., **`POST`**) on the resource.
x-sovd-required-modes	Conditional	Documents the list of required modes which have to be set to an entity in order to invoke the method defined by this Operation object, e.g., writing an ECU configuration.
x-sovd-proximity-proof-required	Conditional	Documents that the execution of an entity operation requires a proof of proximity.
x-sovd-asynchronous-execution	Conditional	Documents whether the execution of the operation is asynchronous or not.
x-sovd-capabilities	Conditional	Documents the list of SOVD capabilities a specific operation of an entity supports at the respective methods available at the SOVD API.

5.2.4 Schema Object

Table 25 provides an overview of all SOVD extension properties for an OpenAPI `Schema` object.

Table 25 Extension Property Schema

Extension Identifier	Usage in SOVD	Description
x-sovd-unit	Conditional	Documents the unit of a property within a JSON schema.
x-sovd-translation-id	Conditional	Documents a translation identifier for a property within a JSON schema, e.g., the name of a resource.

5.2.5 `Header` Object

The following subchapter provides an overview of defined SOVD headers that are used across the SOVD API methods.

5.2.5.1 Authorization Request Header

Whenever a client interacts with resources exposed by an SOVD server by sending HTTP requests to its SOVD API, a valid token shall be provided within the HTTP `Authorization`

header of the request, if access to the resource requires authorization. If the authorization schema requires, other headers may be used as well.

5.2.5.2 Location Response Header

Whenever an SOVD client has to be informed about the resource location of a newly created resource or the client should be re-directed, the resulting response provides a respective HTTP `Location` header where the value is an absolute URI pointing to the related resource.

5.2.5.3 X-SOVD-Mode Request header

If a required ECU mode is not explicitly set before, a Hint-based mode control on the target mode and its value can be provided through the `X-SOVD-Mode` header.

5.2.6 `Parameter` Object

The following subchapter 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. For example, a request for retrieving the set of data of two specific categories and requesting the related schemas in the response will look like the following:

```
GET       {base_uri}/{entity-path}/data?data-categories=identData&data-
categories=currentData&include-schema=true HTTP/1.1
```

5.2.6.1 include-schema

The query parameter `include-schema` allows to specify whether the response should include schema information in form of a `schema` property value or not. The schema attribute will include the definition of all properties in the response. This allows to add OEM-specific extensions. A property may include its own schema property to be more precise on the type definition for the object in which it is contained. If the query parameter include-schema is set to true, the schema properties as defined by the specification is set on all objects in the response.

5.3 Online and Offline Capability Descriptions

5.3.1 Introduction

The following chapter describes how an SOVD client learns about the diagnostic capabilities of a vehicle. 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 a vehicle. For this scenario, the SOVD standard defines an offline capability description (see 5.3.2), which contains complete information about all diagnostic capabilities and for which vehicle variants they are applicable (see 5.4).

In the second scenario, a diagnostic tool developer develops a fully generic diagnostic SOVD client without any prior knowledge about the vehicle. For this case, the SOVD standard defines an online capability description (see 5.3.3), which can be read from the vehicle 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 fully 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 6.4.2).

5.3.2 Offline Capability Description

An offline capability description provides the documentation for all SOVD methods of a vehicle and therefore specifies how to access and interact with all available entities, e.g., ***Component***s or ***App***s.

5.3.3 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 fully compatible to an offline capability description, limited to a single resource. An SOVD server may not provide an online capability description for standardized entities, resources, or resource collections.

5.4 Variant Handling in Offline Capability Description

5.4.1 Introduction

Traditionally, vehicle projects are developed with multiple variants of ECUs which contain 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. For instance, the resource for reading the `SteeringWheelRotationSpeed` can be excluded if the model variant does not support it. The process of handling variants, in classical diagnostics, is achieved through ECU-VARIANTs in ODX data (see [9]).

In SOVD, the two types of entities – ***Components***, which represents the classical ECUs and HPCs -, and ***Apps****,* which represents applications running on HPCs, require the possibility to define variants. The process of handling variants in SOVD can be achieved through an OpenAPI custom extension (see [10]) detailed out in this chapter.

Note: The process to identify and manage variants is proprietary and depends heavily on tools, use cases, engineering processes and rules, etc. of the OEM. Hence, the mechanisms described below are one way of handling variants. There could be alternative mechanisms of managing variants. Such alternatives can be accommodated within the implementations if the OpenAPI and SOVD standard boundaries are not violated.

5.4.2 Extension: x-sovd-applicability

SOVD introduces the custom extension `x-sovd-applicability` that can be defined in the offline capability description. The extension contains the list of variants which support a specific resource (*path*), response structure (*schema*), etc. The specification author can introduce it at various *fields* of the document as needed complying with the OpenAPI standard requirements.

Note: Feature options opted by the customer while purchasing the vehicle also result in numerous variants of the vehicle. They are typically represented in vehicle manufactures using “option codes” which is an expression representing all possible options of the specific vehicle. A support for a resource could also be determined based on that expression. OEMs can add such option code strings to the extension as a bespoke requirement. The processing of such expressions is usually proprietary to the vehicle manufactures and hence not part of this standard.

5.4.3 Usage of x-sovd-applicability by the SOVD Client

A client shall use the information in the offline capability description to drive the behavior of the application based on the variant mounted in the vehicle. The variant information is present as part of the response of the method **`GET /{entity-path}/{entity-Id}`**. Once the variant information is known, the client can use this information to determine, from the `x-sovd-applicability` extension in offline capability description, which resources or schemas match for the vehicle to be diagnosed.

In the example, the resource **`/components/PowerSteering/data/SteeringWheelRotationSpeed`** is applicable for the variant “Pow_Str_Variant_High” only.

```
"/components/PowerSteering/data/SteeringWheelRotationSpeed": {
      "get": {
        "summary": "get the steering wheel rotation speed for
                    diagnostics",
        "description": "get the steering wheel rotation speed for
                        diagnostics",
        "responses": {
          "200": {
            "description": "positive response schema for steering wheel
                            rotation speed",
            "content": {
              "application/json": {
              "schema":{
              "$ref":"#/components/schemas/SteeringWheelRotationSpeed"
                }
              }
            }
          }
        }
      },
      "x-sovd-applicability": {
        "description": "Applicability Extension to specify which variant
                        supports this schema",
        "variant_id_list": {
          "type": "array",
          "items": [
            "Pow_Str_Variant_High"
          ]
```

```
            }
        }
    }
```

6 SOVD REST API

6.1 Introduction

In the following chapters, the standardized SOVD API methods are introduced. 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 provides additional request parameters, the functionality shall be guaranteed also in case they are not used by an SOVD Client.

6.2 Definition of Conventions for API Methods

6.2.1 Query Parameters

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

6.2.2 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.

6.3 Timeout Definitions

The SOVD API definition does not define any 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 has to provide the timeout in seconds then as part of the response which initiated the action.

Table 26 Timeout Definition Header

Header	Operation	Description
`X-SOVD-Status-Retention-Timeout`	**`POST /{entity-path}/operations/{operation-id}/executions`** **`PUT /updates/{update-package-id}/[automated] ||[prepare]|| [execute]`**	Time after which the status information (e.g., execution of an operation or update) will not be available anymore after the action completed.

6.4 API Methods for Access to Capability Description Content

6.4.1 Introduction

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 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 ECU, 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 such as the supported capabilities of the operation or `translation_id`s for parameters within requests or responses.

6.4.2 Query an 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 and its direct child elements as defined by the SOVD standard.
An online capability description is requested for a data resource **`/{entity-path}/data/{data-id}`** via **`GET /{entity-path}/data/{data-id}/docs`**. For example, to enrich the responses with additional details such as the underlying schema of the response using the "`include-schema`" query parameter on the request. Moreover, for each supported HTTP method all defined HTTP response status codes are documented together with the data model of the resulting response payloads.

Path Parameters:

The path parameters depend on the resource to interact with.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Body:

The online capability description of the underlying resource in form of an OpenAPI specification in JSON document format.

Example:

Request:

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

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[OpenAPI Spec]
```

6.5 API Methods for Discovering of Entities and Resources

6.5.1 Introduction

This chapter describes methods to

- Discover entities
- Discover subordinate entities of an entity
- Discover related entities of an entity (e.g., ***Component***s of an ***Area***, ***App***s of a ***Component***)

This allows the client to traverse the topology to find out, which methods can be used for which resources. Using these methods, an SOVD client can traverse the topology to identify entities providing resources.

6.5.2 Query of Available Entities Under This SOVD Server

6.5.2.1 Discover Contained Entities

Method:

```
GET /{Entity-Collection}
```

Method Description:

This method provides the list of contained entities for each requested entity collection (see Table 2).

Note: The collections contain the list of entities the SOVD server knows. It does not reflect the communication status with each entity.

Path Parameters:

Table 27 Path Parameters - Discover contained entities

Parameter Name	Type	Convention	Description
`Entity-Collection`	EntityCollectionType	M	Entity collection type

Query Parameters:

Table 28 Query Parameters - Discover contained entities

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 29 Response Status Codes - Discover contained entities

Status Code	Response Body	Description
`200`	Table 30	The request was successful.

Response Body:

Table 30 Response Body - Discover contained entities of entity collections

Attribute	Type	Convention	Description
`items`	EntityReference`[]`	M	Array of entity references.
`schema`	`OpenAPI Schema`	C	Schema of the contained entities. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 31 `EntityReference` type

Attribute	Type	Convention	Description
`id`	`string`	M	Identifier of the entity.
`name`	`string`	M	Name of the entity.
`translation_id`	`string`	O	Identifier for translating the name
`href`	`string:uri-reference`	M	URI of the subordinate entity including {base_uri}.

Example 1:

Request:

```
GET {base_uri}/components HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "id": "DrivingComputer",
      "name": "Driving computer",
      "href": "{base_uri}/components/DrivingComputer"
    },
    {
      "id":   "Camera",
      "name": "Camera Unit",
      "href": "{base_uri}/components/Camera"
    },
    {
      "id":   "PowerSteering",
      "name": "Power steering Unit",
      "href": "{base_uri}/components/PowerSteering"
    }
  ]
}
```

Example 2:

Request:

```
GET {base_uri}/areas HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "name": "Autonomous Driving",
      "href": "{base_uri}/areas/Driving"
    }
  ]
}
```

6.5.2.2 Query Sub-Entities of an Entity

Method:

```
GET /areas/{area-id}/subareas
GET /components/{component-id}/subcomponents
```

Method Description:

This method provides the list of sub-entities for each ***Area*** or ***Component***. In case of subcomponents or subareas these methods can be used to discover the resource tree.

Path Parameters:

This method accepts either the `area-id` or the `component-id` as a path parameter depending on the requested resource type (***Area*** or ***Component***), which is part of the URI.

Query Parameters:

Table 32 Query Parameters - Query sub-entities of an entity

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 33 Response Status Codes - Query sub-entities of an entity

Status Code	Response Body	Description
`200`	Table 34	The request was successful.

Response Body:

Table 34 Response Body - Query sub-entities of an entity

Attribute	Type	Convention	Description
`items`	EntityReference`[]`	M	Array of entity references.
`schema`	`OpenAPI Schema`	C	Schema of the contained entities. Condition: Only provided if the query parameter `include-schema` is `true`.

Example 1:

Request:

```
GET {base_uri}/components/DrivingComputer/subcomponents HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "id":   "Linux",
      "name": "Linux-based ADAS",
      "href": "{base_uri}/components/DrivingComputer/subcomponents
              /Linux"
    },
    {
      "id":   "AdaptiveAUTOSAR",
      "name": "Adaptive AUTOSAR",
      "href": "{base_uri}/components/DrivingComputer/subcomponents
              /AdaptiveAUTOSAR"
    }
  ]
}
```

Example 2:

Request:

```
GET {base_uri}/areas/Driving/subareas HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "id":   "Perception",
      "name": "Perception system",
      "href": "{base_uri}/areas/Driving/subareas/Perception"
    },
    {
      "id":   "Controlling",
      "name": "Controlling system",
      "href": "{base_uri}/areas/Driving/subareas/Controlling"
    }
  ]
}
```

6.5.2.3 Query Related Entities of an Entity

Method:

```
GET /areas/{area-id}/related-components
GET /components/{component-id}/related-apps
```

Method Description:

This method provides the list of related entities (i.e., ***Component***s of an ***Area*** or ***App***s present on a ***Component***) for each ***Area*** or ***Component***.

Path Parameters:

This method accepts either the `area-id` or the `component-id` as a path parameter depending on the requested resource type (***Area*** or ***Component***), which is part of the URI.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 35 Response Status Codes - Query related entities of an entity

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

Response Body:

Table 36 Response Body - Query related entities of an entity

Attribute	Type	Convention	Description
`items`	EntityReference`[]`	M	Array of entity references.

Example:

Request:

```
GET {base_uri}/areas/Driving/related-components HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "id":   "DrivingComputer",
      "name": "Driving computer",
      "href": "{base_uri}/components/DrivingComputer"
    }
  ]
}
```

6.5.3 Query Entity Capabilities

Method:

```
GET /{entity-path}/{entity-id}
```

Method Description:

This method returns the capabilities of an entity.

Path Parameters:

This method accepts the `entity-id` as a path parameter.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 37 Response Status Codes - Query of entity capabilities

Status Code	Response Body	Description
`200`	Table 38	The request was successful.

Response Body:

The response body contains a property for each supported resource and related collection (see 4.3).

Note: This method returns the resource and related collections. Querying the resources of a specific resource-collection is described in the individual method subchapters.

Table 38 Response Body - Query of entity capabilities

Attribute	Type	Convention	Description
`id`	`string`	M	Entity-id
`name`	`string`	M	Name of the entity
`translation_id`	`string`	O	Identifier for translating the name
`variant`	`AnyValue`	C_3	Identification of the variant
`configurations`	`string:uri-reference`	C_1	A reference to the configurations collection
`bulk-data`	`string:uri-reference`	C_1	A reference to the bulk data collection
`data`	`string:uri-reference`	C_1	A reference to the data collection
`data-lists`	`string:uri-reference`	C_1	A reference to the data-lists collection
`faults`	`string:uri-reference`	C_1	A reference to the faults collection

operations	string:uri-reference	C_1	A reference to the operations collection
updates	string:uri-reference	C_1	A reference to the updates collection
modes	string:uri-reference	C_1	A reference to the modes collection
relatedapps	string:uri-reference	C_2	A reference to the reference-collection providing the ***App***s (only available for ***Component***s)
related components	string:uri-reference	C_2	A reference to the reference-collection providing the ***Component***s (only available for ***Area***s)
subareas	string:uri-reference	C_2	A reference to the reference-collection providing the subareas (only available for ***Area***s)
subcomponents	string:uri-reference	C_2	A reference to the reference-collection providing the subcomponents (only available for ***Component***s)
locks	string:uri-reference	C_1	A reference to the reference-collection locks
logs	string:uri-reference	C_1	A reference to the resource logs

- C_1: Reference is only present if the entity supports this resource collection or resource.
- C_2: Reference is only present if the collections are not empty.
- C_3: The attribute is provided in case the SOVD server supports variant identification as described in 5.4.

Example

Request:

```
GET {base_uri}/apps/AdvancedLaneKeeping HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "id":             "AdvancedLaneKeeping",
  "name":           "Advanced Lane Keeping",
  "configurations": "{base_uri}/apps/AdvancedLaneKeeping/
                     configurations",
  "bulk-data":      "{base_uri}/apps/AdvancedLaneKeeping/bulk-data",
  "data":           "{base_uri}/apps/AdvancedLaneKeeping/data",
```

```
    "faults":          "{base_uri}/apps/AdvancedLaneKeeping/faults",
    "operations":      "{base_uri}/apps/AdvancedLaneKeeping/operations",
    "logs":            "{base_uri}/apps/AdvancedLaneKeeping/logs"
}
```

6.6 API Methods for Fault Handling

6.6.1 Introduction

The SOVD API provides methods for reading and deleting faults from an entity. A fault resource represents any kind of identified failure. In addition, for a single fault more detailed information (e.g., environment data from UDS) can be retrieved. If supported by the entity, single fault entries may be deleted as well. The method for reading faults provides sophisticated filtering mechanisms like providing only fault entries which fulfill certain criteria such as being an active fault.

The provided mechanisms are flexible enough to support reading fault entries from HPCs and ECUs. The provided mechanisms allow the transfer of additional contextual information for analyzing the root cause of the fault.

6.6.2 Read Faults from an Entity

Method:

`GET /{entity-path}/faults`

Method Description:

This method provides the fault entries which are detected for an entity. The returned fault entries may be filtered by their `status`, `severity` or `scope` using query parameters.

For the faults resource collection (`/{entity-path}/faults`) the capability description contains the schema definition of this collection and therefore not a detailed description of each fault.
The fault entries are contained in `/{entity-path}/faults/{fault-code}` and the capability description of each fault contains the details like the schema of the `environment_data`.

Path Parameters:

This method does not support path parameters.

Query Parameters:

If multiple filter parameters are included they are combined by logically AND.

Table 39 Query Parameters - Read faults from an entity

Parameter Name	Type	Convention	Description
`status[key]`	`string`	O	Filters the available elements based on a `key` from the status. The

			provided value is a full match against the `key` and value of the attribute `status` in the response. For filtering by multiple status, the parameter is repeated (0 .. *). If multiple status are provided, they are logically OR combined. The `key`s supported for faults are defined in the capability description of the resource collection faults.
`severity`	`integer`	O	Filter fault entries by their severity. Any fault with a severity equal to or below the given number will be included in the response.
`scope`	`string`	O	The scope (e.g., user-defined fault memories) for which fault entries are retrieved. The capability description defines which scopes are supported.
`include-schema`	`boolean`	O	Specifies whether the response should include schema information or not. The default is `false`.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 40 Response Status Codes - Read faults from an entity

Status Code	Response Body	Description
`200`	Table 41	The request was successful.

Response Body:

Table 41 Response Body - Read faults from an entity

Attribute	Type	Convention	Description
`items`	Fault[]	M	Collection of faults set in the entity
`schema`	`OpenAPI Schema`	C	Schema for describing the fault response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 42 `Fault` type

Attribute	Type	Convention	Description
`code`	`string`	M	Fault code in the native representation of the entity.
`scope`	`string`	O	Defines the scope (e.g., user-defined fault memories) for fault entries. The capability description defines which scopes are supported.
`display_code`	`string`	O	Display representation for the fault `code`.
`fault_name`	`string`	M	Name / description of the fault
`fault_translation_id`	`string`	O	Optional identifier for translating the name
`severity`	`integer`	O	Severity defines the impact on the availability of the vehicle. Lower number means higher severity. For a classic ECU this resembles the level from ODX, for HPCs the SOVD API recommends 1 = `FATAL`, 2 = `ERROR`, 3 = `WARN`, and 4 = `INFO`.
`status`	`object`	C_1	Detailed status information for the fault as `key` value pairs. For a classic ECU this resembles the status byte of a DTC. The content of the keys and values is OEM-specific. The keys are defined in the capability description. See Notes below.
`symptom`	`string`	C_1	Detailed symptom / failure mode information for the fault. For classic ECU this resembles the symptom of a DTC. The content of the attribute is OEM-specific.
`symptom_translation_id`	`string`	C_1	Translation identifier for the symptom name.
`schema`	`OpenAPI Schema`	C_2	Schema of the fault element.

- C_1: This attribute is only present in case the fault provides the requested information.
- C_2: This attribute is only present in case the query attribute `include-schema` is `true`.

Note: The status can be aggregated in the key `aggregatedStatus` and is then available for HPCs and classic ECUs. The aggregation of the status is vendor specific.

Note: For UDS based ECUs, the nomenclature of ISO 14229-1[1] as defined in appendix D.2.3 is used as keys for the `status` attribute. In addition, the status byte in hex is included in the key `mask`.

Note: For HPCs it is advised to follow the nomenclature of ISO 14229-1 for status keys if reasonable.

Example (Classic ECU):

Request:

```
GET
{base_uri}/components/Camera/faults?status[aggregatedStatus]=active
HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "code": "0012E3",
      "scope": "Default",
      "display_code": "P102",
      "fault_name": "No signal from sensor",
      "fault_translation_id": "CAMERA_0012E3_tid",
      "severity": 1,
      "status":
        {
          "testFailed": "1",
          "testFailedThisOperationCycle": "1",
          "pendingDTC": "1",
          "confirmedDTC": "1",
          "testNotCompletedSinceLastClear": "0",
          "testFailedSinceLastClear": "1",
          "testNotCompletedThisOperationCycle": "0",
          "warningIndicatorRequested": "0",
          "aggregatedStatus": "active"
        }
    },
    {
      "code": "003C01",
      "scope": "Default",
      "display_code": "P03C",
      "fault_name": "Battery voltage low",
      "fault_translation_id": "Low Vbat_tid",
      "severity": 2,
      "status":
        {
          "testFailed": "1",
          "testFailedThisOperationCycle": "1",
          "pendingDTC": "1",
          "confirmedDTC": "1",
          "testNotCompletedSinceLastClear": "0",
          "testFailedSinceLastClear": "1",
          "testNotCompletedThisOperationCycle": "0",
```

```
            "warningIndicatorRequested": "0",
            "aggregatedStatus": "active"
          }
      }
    ]
  }
```


Example (HPC):

Request:

```
GET {base_uri}/apps/AdvancedLaneKeeping/faults?severity=1 HTTP/1.1
```

Response:


```
HTTP/1.1 200 OK
{
  "items": [
    {
      "code": "modelMissing",
      "scope": "Default",
      "fault_name": "No Object Recognition Model available",
      "fault_translation_id": "ALK NoObjModel_tid",
      "severity": 1,
      "status":
        {
          "aggregatedStatus": "active"
        }
    }
  ]
}
```


Further examples for requests:

- UDS Status byte mnemonic:

```
GET {base_uri}/{entity-
path}/faults?status[pendingDTC]=1&status[confirmedDTC]=1 HTTP/1.1
```

- UDS Status mask:

```
GET {base_uri}/{entity-path}/faults?status[mask]=1A HTTP/1.1
```

- Interpreted UDS status byte:

```
GET {base_uri}/{entity-path}/faults?status[aggregatedStatus]=active
HTTP/1.1
```

- Faults from crashed applications:

```
GET {base_uri}/{entity-path}/faults?status[state]=crashed HTTP/1.1
```

6.6.3 Read Details for a Fault

Method:

`GET /{entity-path}/faults/{fault-code}`

Method Description:

This method provides the details for a fault-code.

Path Parameters:

Table 43 Path Parameters - Read details for a fault

Parameter Name	Type	Convention	Description
`fault-code`	`string`	M	The fault code for which the detailed information should be retrieved. The value shall be either returned by the request `GET /{entity-path}/faults` or defined in the capability description.

Query Parameters:

Table 44 Query Parameters - Read details for a fault

Parameter Name	Type	Convention	Description
`include-schema`	`boolean`	O	Specifies whether the response should include the schema information for the response. The default is `false`.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 45 Response Status Codes - Read details for a fault

Status Code	Response Body	Description
`200`	Table 46	The request was successful.

Response Body:

Table 46 Response Body - Read details for a fault

Attribute	Type	Convention	Description
`item`	Fault	M	Fault description
`environment_data`	`AnyValue`	C	Additional OEM-specific information describing the environment of the system at the time of setting the fault. Condition: Only set if the fault provides the requested information.

errors	DataError[]	C	Condition: Only set, if the value environment_data represents an error, and this attribute describes the error more precisely.
schema	OpenAPI Schema	C	Schema for describing the response. Condition: Only provided if the query parameter include-schema is true.

Example:

Request:

```
GET {base_uri}/apps/AdvancedLaneKeeping/faults/0012E3 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "item":
    {
      "code": "0012E3",
      "scope": "Default",
      "display_code": "P102",
      "fault_name": "No signal from sensor",
      "fault_translation_id": "CAMERA_0012E3_tid",
      "severity": 1,
      "status":
        {
          "testFailed": "1",
          "testFailedThisOperationCycle": "1",
          "pendingDTC": "0",
          "confirmedDTC": "1",
          "testNotCompletedSinceLastClear": "1",
          "testFailedSinceLastClear": "0",
          "testNotCompletedThisOperationCycle": "0",
          "warningIndicatorRequested": "0",
          "mask": "1E",
          "aggregatedStatus": "active"
        }
    },
  "environment_data":
    {
      "id": "env-data",
      "data": {
        "battery_voltage": 12.8,
        "occurence_counter": 12,
        "first_occurence": "2021-06-22T11:47:22Z",
        "last_occurence": "2021-07-01T10:12:53Z"
      }
    }
}
```

6.6.4 Delete All Faults of an Entity

Method:

`DELETE /{entity-path}/faults`

Method Description:

This method deletes all fault entries of an entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 47 Query Parameters - Delete all faults of an entity

Parameter Name	Type	Convention	Description
`scope`	`string`	O	Defines the scope (e.g., user-defined fault memories) for which fault entries are deleted. The capability description defines which scopes are supported.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 48 Response Status Codes - Delete all faults of an entity

Status Code	Response Body	Description
`204`	`--`	The request was successful.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE {base_uri}/components/Camera/faults HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.6.5 Delete Single Fault of an Entity

Method:

```
DELETE /{entity-path}/faults/{fault-code}
```

Method Description:

This method deletes the given fault entry from an entity.

Path Parameters:

Table 49 Path Parameters - Delete single fault of an entity

Parameter Name	Type	Convention	Description
`fault-code`	`string`	M	The fault code which should be deleted. The value shall be either returned by the request `GET /{entity-path}/faults` or defined in the capability description.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 50 Response Status Codes - Delete single fault of an entity

Status Code	Response Body	Description
`204`	`--`	Fault entry has been deleted successfully.
`404`	`--`	The SOVD client tried to delete a fault entry which is not defined for the entity.
`409`	GenericError	The SOVD client does not fulfill the conditions for deleting a single DTC, e.g., the DTC is safety relevant. The SOVD server should supply information in the `GenericError` which condition is not met (e.g., safety conditions, DTC not set, …)
`501`	`--`	The request for deleting single fault entries is not supported by the entity.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE {base_uri}/apps/AdvancedLaneKeeping/faults/modelMissing HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.7 API Methods for Data Resource Read / Write Access

6.7.1 Introduction

The SOVD API described in this subchapter allows an SOVD client to read and write various kinds of data values from and to an entity. SOVD does not differentiate between identification, measurements, and other data values. To allow an SOVD client to differ between various kinds of data values, the categories defined by SOVD can be used to distinguish between measurements etc.

Table 51 `DataCategory` type

Category	Description
identData	Identifications Read access to fixed parameters that identify entities within a vehicle, e.g., part number or VIN.
currentData	Measurements Read access to dynamically changing values, e.g., battery voltage.
storedData	Parameters Read and write access to parameters in the vehicle.
sysInfo	System information Read access to dynamically changing system resources, e.g., CPU load.

Note: OEM-specific categories can be included using alpha-numeric characters, hyphens, and underscores and have to start with **`x-<oem>`**. An implementor of the SOVD API may introduce additional categories. These categories are then not supported by an off-the-shelf SOVD client.

The SOVD server optionally supports the grouping of data values for data which belongs together. Groups are OEM specific.

SOVD clients having to read multiple values at once multiple times can define a temporary data list for this. By reading the data list, the SOVD client will get all defined values in one response.

6.7.2 Query of Data Resource Categories and Groups

6.7.2.1 Retrieve Categories Supported by a Data Resource Collection

Method:

```
GET /{entity-path}/data-categories
```

Method Description:

This method returns the data categories provided by an entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 52 Response Status Codes - Retrieve categories supported by a data resource collection

Status Code	Response Body	Description
200	Table 53	The request was successful.

Response Body:

Table 53 Response Body - Retrieve categories supported by a data resource collection

Attribute	Type	Convention	Description
items	DataCategory []	M	Supported categories.

Example:

Request:

```
GET {base_uri}/apps/WindowControl/data-categories HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    "currentData",
    "identData"
  ]
}
```

6.7.2.2 Retrieve Groups Supported by a Data Resource Collection

Method:

```
GET /{entity-path}/data-groups
```

Method Description:

This method provides the groups defined for an entity. Groups are associated with a category.
The concept of groups is not intended to be used for partitioning data from different subcomponents, e.g., LIN slaves should be mapped to subcomponents of the master ECU and are thus accessible using a URI like {base_uri}/**components/master/subcomponents/lin-slave**.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 54 Response Status Codes - Retrieve groups supported by a data resource collection

Status Code	Response Body	Description
200	Table 55	The request was successful.

Response Body:

Table 55 Response Body - Retrieve groups supported by a data resource collection

Attribute	Type	Convention	Description
items	ValueGroup[]	M	Available groups.

Table 56 `ValueGroup` type

Attribute	Type	Convention	Description
id	string	M	Unique identifier for a group across all categories
category	DataCategory	M	Category for which the group is defined
category_translation_id	string	O	Identifier for translating the category name

group	string	O	Name of the group
group_ translation_id	string	O	Identifier for translating the group name

Example:

Request:

```
GET {base_uri}/apps/WindowControl/data-groups HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "id": "front",
      "category": "currentData"
    },
    {
      "id": "rear",
      "category": "currentData"
    }
  ]
}
```

6.7.3 Retrieve List of All Data Provided by the Entity

Method:

GET /{entity-path}/data

Method Description:

This method provides the list of all data resources available for an entity. The returned data may be filtered by their category (e.g., to limit the list to current data values) or by their group identifiers.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 57 Query Parameters - Retrieve list of all data provided by the entity

Parameter Name	Type	Convention	Description
groups	string[]	C	Filters the available elements based on the groups they belong to. If an element belongs to multiple groups, it is included if one of the groups is part of these multiple groups. If a category is defined, the categories parameter is ignored by the SOVD server. The groups

			parameter takes precedence over the category if an SOVD client defines both as the `groups` is more restrictive.
`categories`	DataCategory `[]`	C	Filters the available elements based on their category. If `groups` are defined, the `categories` parameter is ignored by the SOVD server.
`include-schema`	`boolean`	O	Specifies whether the response should include schema information or not. The default is `false`.

An SOVD client should either provide the groups or categories query parameter in the request, but not both.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 58 Response Status Codes - Retrieve list of all data provided by the entity

Status Code	Response Body	Description
`200`	Table 59	The request was successful.

Response Body:

Table 59 Response Body - Retrieve list of all data provided by the entity

Attribute	Type	Convention	Description
`items`	ValueMetaData `[]`	M	Definition of the data.
`schema`	`OpenAPI Schema`	C	The schema definition of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 60 `ValueMetaData type`

Attribute	Type	Convention	Description
`id`	`string`	M	Unique identifier for a value
`name`	`string`	M	Name of the data value

translation_id	string	O	Identifier for translating the data's name
category	DataCategory	M	Category of the data value
groups	string[]	O	The identifiers of the groups to which the resource belongs to.

Example:

Request:

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

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "id":       "DriverWindow",
      "name":     "Position of driver window",
      "category": "currentData",
      "groups": [ "front" ]
    },
    {
      "id":       "PassengerWindow",
      "name":     "Position of passenger window",
      "category": "currentData",
      "groups": [ "front" ]
    },
    {
      "id":       "RearWindows",
      "name":     "Position of rear windows",
      "category": "currentData",
      "groups": [ "rear" ]
    },
    {
      "id":       "AppInfo",
      "name":     "Window Control Version Numbers",
      "category": "identData"
    }
  ]
}
```

6.7.4 Read Single Data Value from an Entity

Method:

GET /{entity-path}/data/{data-id}

Method Description:

This method retrieves the value of a single data resource, like VIN, battery voltage, or CPU load. The response includes all values associated with the resource and cannot be filtered down to individual attributes of the resource.

Path Parameters:

Table 61 Path Parameters - Read single data value from an entity

Parameter Name	Type	Convention	Description
`data-id`	`string`	M	The identifier for the value to be read

Query Parameters:

Table 62 Query Parameters - Read single data value from an entity

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 63 Response Status Codes - Read single data value from an entity

Status Code	Response Body	Description
`200`	Table 64	The request was successful.

Response Body:

Table 64 `ReadValue` type

Attribute	Type	Convention	Description
`id`	`string`	M	Unique identifier of a value.
`data`	`AnyValue`	M	The value of the data. The type of the value is defined by the `schema`.
`errors`	DataError`[]`	C	Condition: Only set, if the value `data` represents an error and this attribute describes the error more precisely.
`schema`	`OpenAPI Schema`	C	Schema for the data. Condition: Only provided if the query parameter `include-schema` is `true`.

Note: The attribute `errors` is used if the method execution has been successful, but the underlying resource returned an erroneous value. A common example is that an ECU did not provide the actual sensor value, but an indication that the value from the sensor is not plausible. In this case a status code signaling an error would be inappropriate. This applies also to single parameters inside the response.

Note: Complex data structures (nested parameters, lists) will be described via the respective data type inside the schema.

Example:

Request:

```
GET {base_uri}/apps/WindowControl/data/RearWindows?include-schema=true
HTTP/1.1
```

Response (Success):

```
HTTP/1.1 200 OK

{
  "id": "RearWindows",
  "data": {
    "PositionLeft": 100,
    "PositionRight": 0
  },
  "schema": {
    "type": "object",
    "id": "string",
    "data": {
      "type": "object",
      "properties": {
        "PositionLeft": {
          "type": "integer",
          "format": "int32",
          "minimum": 0,
          "maximum": 100,
          "x-sovd-unit": {
            "display_name": "%"
          }
        },
        "PositionRight": {
          "type": "integer",
          "format": "int32",
          "minimum": 0,
          "maximum": 100,
          "x-sovd-unit": {
            "display_name": "%"
          }
        }
      }
    }
  }
}
```

Response (Value returned by component could not be evaluated):

```
HTTP/1.1 200 OK
{
   "id": "RearWindows",
   "data": {
```

```
      "PositionLeft": 100,
      "PositionRight": 0
    },
    "errors": [
      {
        "path": "/data/PositionRight",
        "error": {
          "error_code": "vendor-specific",
          "vendor_code": "value-constraint-violated",
         "message": "Value provided by component could not be evaluated",
          "parameters": {
            "internal_value": "FF",
            "description": "Signal implausible"
          }
        }
      }
    ]
}
```


Response (Negative response):


```
HTTP/1.1 200 OK
{
    "id": "RearWindows",
    "data": {
    },
    "errors": [
      {
        "path": "/data",
        "error": {
          "error_code": "error-response",
          "message": "Error response received",
          "parameters": {
            "service": 34,
            "nrc": 49
          }
        }
      }
    ]
}
```


6.7.5 Read Multiple Data Values from an Entity

6.7.5.1 Retrieve List of All Data-Lists Provided by the Entity

Method:

GET /{entity-path}/data-lists

Method Description:

Provides the list of all data-list resources available for an entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 65 Response Status Codes - Retrieve list of all data-lists provided by the entity

Status Code	Response Body	Description
200	Table 66	The request was successful.

Response Body:

Table 66 Response Body - Retrieve list of all data-lists provided by the entity

Attribute	Type	Convention	Description
items	DataListEntry []	M	The list of data lists together with the individual data identifiers

Table 67 `DataListEntry` type

Attribute	Type	Convention	Description
id	string	M	Identifier of the data list
items	ValueMetaData []	M	Definition of the data.

Example:

Request:

```
GET {base_uri}/apps/WindowControl/data-lists HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "id": "7ab52f10-5ea9-4610-a1de-c414c8265ce4",
      "items": [
        {
          "id":       "DriverWindow",
          "name":     "Position of driver window",
          "category": "currentData",
          "groups": [ "front" ]
        },
        {
          "id":       "PassengerWindow",
          "name":     "Position of passenger window",
          "category": "currentData",
```

```
            "groups": [ "front" ]
          },
          {
            "id":       "RearWindows",
            "name":     "Position of rear windows",
            "category": "currentData",
            "groups": [ "rear" ]
          }
        ]
      }
    ]
  }
```

6.7.5.2 Creating a Data List for Reading Multiple Data Values at Once from an Entity

Method:

`POST /{entity-path}/data-lists`

Method Description:

This method creates a temporary resource for reading multiple data resources with one request from an entity. The identifiers of the requested data resources of the given entity shall be listed in the request body. The SOVD API does not impose any restrictions on the category or group of the identifier, especially identifiers from different groups and categories maybe mixed.
The resource has to be deleted explicitly by the SOVD client. If the SOVD client does not delete the resource, the SOVD server can delete it after a certain period if the resource is not used anymore or upon SOVD server restart.

Note: This specification does neither forbid nor mandate that an SOVD server has to return a unique identifier if SOVD clients provide the same identifiers in a request.

Any SOVD client may access a data list if the data list identifier is known. If an SOVD client does not have access to some (or all) resources referenced in the data list, the creation of the data list will anyways succeed. However, an SOVD client with insufficient rights will get an error when reading the data list (for details see Table 17).

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

```
Content-Type: application/json
```

Request Body:

Table 68 Request Body - Creating a data list for reading multiple data values at once from an entity

Attribute	Type	Convention	Description
`ids`	`string[]`	M	The data resource identifiers which should be read as part of the data list.

Response Header:

The response includes the `Location` header to redirect the SOVD client to reading the resource for requesting the values:
`Location: {base_uri}/{entity-path}/data-lists/{id}`

Response Status Codes:

Table 69 Response Status Codes - Creating a data list for reading multiple data values at once from an entity

Status Code	Response Body	Description
`201`	Table 70	The SOVD server created the data list.
`501`	`--`	The SOVD server does not support the creation of data list resources
`400`	GenericError	The SOVD client request included a resource identifier which does not exist. The identifiers are listed in the error.

Response Body:

Table 70 Response Body - Creating a data list for reading multiple data values at once from an entity

Parameter Name	Type	Convention	Description
`id`	`string`	M	The identifier which has to be used for querying the values of the data list.

Example:

Request:

```
POST {base_uri}/apps/WindowControl/data-lists HTTP/1.1
Content-Type: application/json

{
  "ids": [
    "DriverWindow",
    "PassengerWindow",
    "RearWindows"
  ]
}
```

Response:

```
HTTP/1.1 201 Created
Location: {base_uri}/apps/WindowControl/data-lists/7ab52f10-5ea9-4610-
a1de-c414c8265ce4

{
  "id": "7ab52f10-5ea9-4610-a1de-c414c8265ce4"
}
```

6.7.5.3 Read Multiple Data Values at Once from an Entity Using a Data List

Method:

`GET /{entity-path}/data-lists/{data-list-id}`

Method Description:

This method returns the values defined by the given data list.

Path Parameters:

Table 71 Path Parameters - Read multiple data values at once from an entity using a data list

Parameter Name	Type	Convention	Description
`data-list-id`	`string`	M	The identifier for the data list returned by the SOVD server upon creation.

Query Parameters:

Table 72 Query Parameters - Read multiple data values at once from an entity using a data list

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 73 Response Status Codes - Read multiple data values at once from an entity using a data list

Status Code	Response Body	Description
200	Table 74	The request was successful.

Response Body:

Table 74 Response Body - Read multiple data values at once from an entity using a data list

Attribute	Type	Convention	Description
items	ReadValue[]	M	The list of values for the data resources specified as part of the data list.
schema	OpenAPI Schema	C	The schema definition of the response. Condition: Only provided if the query parameter include-schema is true.

If an SOVD client is not authorized to read a certain resource, the attribute error_code of the item shall be set to insufficient-access-rights and the path attribute defines the value which could not be read.

Note: The order of the elements is implementation specific and may not be identical to the order of the URIs when the data list has been defined.

Example:

Request:

```
GET {base_uri}/apps/WindowControl/data-lists/7ab52f10-5ea9-4610-a1de-c414c8265ce4 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "id": "7ab52f10-5ea9-4610-a1de-c414c8265ce4",
  "items": [
    {
      "id": "DriverWindow",
      "data": {
        "Position": 100
      }
    },
    {
      "id": "PassengerWindow",
```

```
        "data": {
          "Position": 100
        }
      },
      {
        "id": "RearWindows",
        "data": {
          "PositionLeft": 100,
          "PositionRight": 0
        }
      }
    ]
  }
```


6.7.5.4 Delete an Existing Data List

Method:

`DELETE /{entity-path}/data-lists/{data-list-id}`

Method Description:

This method removes the temporarily defined data list.

Note: Subsequent calls for this data list from other SOVD clients will result in status code 404 afterwards.

Path Parameters:

Table 75 Path Parameters - Delete an existing data list

Parameter Name	Type	Convention	Description
`data-list-id`	`string`	M	The identifier for the data list.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 76 Response Status Codes - Delete an existing data list

Status Code	Response Body	Description
`204`	`--`	The request was successful.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE {base_uri}/apps/WindowControl/data-lists/7ab52f10-5ea9-4610-a1de-c414c8265ce4 HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.7.6 Write a Data Value to an Entity

Method:

`PUT /<entity-path>/data/{data-id}`

Method Description:

This method writes the value of a data resource of an entity.

Note: Only a single resource can be written.

Path Parameters:

Table 77 Path Parameters - Write a data value data to an entity

Parameter Name	Type	Convention	Description
`data-id`	`string`	M	The ID for the value to be written

Query Parameters:

This method does not support query parameters.

Request Headers:

```
Content-Type: application/json
```

Request Body:

Table 78 Request Body - Write a data value data to an entity

Attribute	Type	Convention	Description
`data`	`AnyValue`	M	The value of the data. The type of the value is defined by the resource.
`signature`	`string`	O	Additional security artefact for the data itself. This is independent from the authentication in the method header.

Response Status Codes:

Table 79 Response Status Codes - Write a data value data to an entity

Status Code	Response Body	Description
204	--	Data written
400	DataError	The request body does either not contain all required parameter values or the signature for the provided values is wrong.

Response Body:

This method does not provide a response body.

Example:

Request:

```
PUT {base_uri}/apps/WindowControl/data/AppInfo
Content-Type: application/json HTTP/1.1

{
  "data":
    {
      "version": "1.15.0",
      "vendor":  "TheWindowLifterCompany"
    }
}
```

Response:

```
HTTP/1.1 204 No Content
```

6.8 API Methods for Configuration

6.8.1 Introduction

Components and ***App***s need to be configured to a specific environment, e.g., vehicle equipment, country, customer demand, variant coding, etc.

To achieve this goal, the SOVD API supports methods for reading and writing of configuration data. A configuration resource can be expressed as a set of parameters or as a byte stream to support multiple types of configuration data e.g., manifest (e.g., ASCII) or binary files.

6.8.2 Retrieve List of All Configurations Provided by the Entity

Method:

```
GET /{entity-path}/configurations
```

Method Description:

Provides the configurations provided by an entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 80 Query Parameters - Retrieve list of all configurations provided by the entity

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 81 Response Status Codes - Retrieve list of all configurations provided by the entity

Status Code	Response Body	Description
`200`	Table 82	The request was successful.

Response Body:

Table 82 Response Body - Retrieve list of all configurations provided by the entity

Attribute	Type	Convention	Description
`items`	ConfigurationMetaData `[]`	M	The list of all configurations provided by the entity.
`schema`	`OpenAPI Schema`	C	The schema definition of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 83 `ConfigurationMetaData` type

Attribute	Type	Convention	Description
`id`	`string`	M	Unique identifier of a configuration data
`name`	`string`	M	Name of the configuration data
`translation_id`	`string`	O	Identifier for translating the configurations data name
`type`	ConfigurationType	M	Configuration type definition
`version`	`string`	C_1	Version of the configuration data
`content_type`	`string`	C_1	MIME type with information for the SOVD client about the content type, e.g., XML etc.

Table 84 `ConfigurationType`

Value	Description
`bulk`	The configuration data is provided as bulk data and the attributes `version` and `content_type` define how it is interpreted.
`parameter`	The configuration is provided as single attributes of a JSON object.

- C_1: `version` and `content_type` are mandatory for bulk data configurations but are not present for parameter-based configurations.

Example

Request:

```
GET {base_uri}/apps/AdvancedLaneKeeping/configurations HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "id": "ALKConfig",
      "name": "Configuration for the Advanced Lane Keeping System",
      "type": "parameter",
    },
    {
      "id": "ObjectRecognitionModel",
      "name": "Model of objects to be recognized",
      "type": "bulk",
      "version": "1.45.2107",
      "content_type": "application/octet-stream"
    }
  ]
}
```

6.8.3 Read Configuration

6.8.3.1 Introduction

The SOVD API provides a method for reading configuration data from an entity. A configuration can either be read as bulk data or as parameters. Either way, a configuration is always read as a whole, since it is a single resource. E.g., it is not possible to limit the access to certain configuration parameters of a resource. Available `configuration-id`s are requested with Retrieve List of All Configurations Provided by the Entity.

Method:

```
GET /{entity-path}/configurations/{configuration-id}
```

Method Description:

This method is used to read a configuration from an entity, which can either be

- bulk data or
- parameter data (MIME type: `application/json`)

The SOVD standard follows the server-driven content negotiation where the client defines its preferences using the Accept request header. If no Accept request header is provided, the SOVD server returns a default representation.

Path Parameters:

Table 85 Path Parameters - Read configuration

Parameter Name	Type	Convention	Description
`configuration-id`	`string`	M	The identifier for the configuration to be read.

6.8.3.2 Read Configuration as Bulk Data

This method only specifies the interface for reading bulk data, i.e., the requested MIME type is not `application/json`. It does not imply a certain implementation. Bulk data typically uses the `Content-Type application/octet-stream`. However, an SOVD client or SOVD server may always use a more specific MIME type than `application/octet-stream` if applicable.
The method can use the existing bulk data handling mechanism as described in subchapter 6.13.

Query Parameters:

This method does not support query parameters.

Request Headers:

`Accept: application/octet-stream` (or more specific)

Response Status Codes:

Table 86 Response Status Codes - Read Configuration as Bulk Data

Status Code	Response Body	Description
200	Response Body	The request was successful.
406	--	The configuration cannot be provided in the requested MIME type.

Response Body:

The content of the response body is OEM specific. Depending on the implementation, the transferred bulk data can use `multipart/signed`, `multipart/form-data` with security parameters in a JSON object, or simply `application/octet-stream` (if it is not confidential information).

Example

Request:

```
GET
{base_uri}/apps/AdvancedLaneKeeping/configurations/ObjectRecognitionMo
del HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

Content-Type: multipart/form-data; boundary=formdataboundary
--formdataboundary
Content-Disposition: form-data; name="signature"
Content-Type: application/json
{
"signature": "84e0982a082024fac464bc5eb69c0745",
}
--formdataboundary
Content-Disposition: form-data; name="ObjectRecognitionModel";
Content-Type: application/octet-stream
[4D617920746865206F726365206265207769746820796F75210D0A]
--formdataboundary
```

6.8.3.3 Read Configuration as Parameters

This method reads the configuration as parameter values from an entity, e.g., reading configuration of an entity as a list of key-value pairs (APP settings) or ECU config files.

Query Parameters:

Table 87 Query Parameters - Read configuration as parameters

Parameter Name	Type	Convention	Description
include-schema	boolean	O	Specifies whether the response should include schema information or not.

			The default is `false`.

Request Headers:

The SOVD server expects that the `Accept` header requests the `Content-Type` `application/json`.

Response Status Codes:

Table 88 Response Status Codes - Read configuration as parameters

Status Code	Response Body	Description
`200`	ReadValue	The request was successful.
`406`	--	The configuration cannot be provided as parameter data, i.e., the configuration cannot be provided as JSON.

Response Body:

The Response Body contains the object of type ReadValue.

Example

Request:

```
GET {base_uri}/
apps/AdvancedLaneKeeping/configurations/ALKConfig?include-schema=true
Accept: application/json HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "id": "ALKConfig",
  "data": {
    "MaximumSpeed": 150,
    "MinimumDistanceToLine": 15.0,
    "EmergencyLaneDistanceToLine": 5.0
  },
  "schema": {
    "type": "object",
    "properties": {
      "MaximumSpeed": {
        "type": "integer",
        "format": "int32",
        "minimum": 0,
        "maximum": 250,
        "x-sovd-unit": {
          "display_name": "km/h",
          "physical_dimension":
            {
              "length": 1,
              "time": -1
            }
        }
      },
      "MinimumDistanceToLine": {
```

```
          "type": "number",
          "format": "float",
          "minimum": 10.0,
          "maximum": 20.0,
          "x-sovd-unit": {
            "display_name": "cm",
            "physical_dimension":
              {
                "length": 1
              }
          }
        },
        "EmergencyLaneDistanceToLine": {
          "type": "number",
          "format": "float",
          "minimum": 5.0,
          "maximum": 10.0,
          "x-sovd-unit": {
            "display_name": "cm",
            "physical_dimension":
              {
                "length": 1
              }
          }
        }
      }
    }
  }
```


6.8.4 Write Configuration

6.8.4.1 Introduction

The SOVD API provides a method for writing configuration data to an entity. A configuration can either be written as bulk data or as parameters. Either way, a configuration is always written as a whole, since it is a single resource. E.g., it is not possible to write certain configuration parameters individually without providing the values for the other configuration parameters of the resource as well.

Method:

`PUT /{entity-path}/configurations/{configuration-id}`

Method Description:

This method is used to write a configuration to an entity, which can either be

- bulk data, or
- parameter data (MIME type: `application/json`)

The SOVD client shall tell the SOVD server in the `Content-Type` header, which MIME type is used for the data in the request body.

Path Parameters:

Table 89 Path Parameters - Write configuration

Parameter Name	Type	Convention	Description
`configuration-id`	`string`	M	The identifier for the configuration to be written

6.8.4.2 Write Configuration as Bulk Data

This method only specifies the interface for writing bulk data, i.e., the `Content-Type` is not `application/json`. It does not imply a certain implementation. However, it can make use of the existing bulk data handling mechanism as described in subchapter 6.13.

Query Parameters:

This method does not support query parameters.

Request Headers:

`Content-Type: application/octet-stream` (or more specific)

Request Body:

The content of the request body is OEM specific. Depending on the implementation, the transferred bulk data can use `multipart/signed`, `multipart/form-data` with security parameters in a JSON object, or simply `application/octet-stream` (if it is not confidential information).

Response Status Codes:

Table 90 Response Status Codes - Write Configuration as Bulk Data

Status Code	Response Body	Description
`204`	`--`	Configuration written
`409`	GenericError	Unable to write configuration due to unsatisfied precondition. E.g., after satisfying precondition “vehicle not in motion” this should be possible.

Response Body:

This method does not provide a response body.

Example

Request:

```
PUT {base_uri}/apps/AdvancedLaneKeeping/configurations/
ObjectRecognitionModel HTTP/1.1

Content-Type: multipart/form-data; boundary=formdataboundary
--formdataboundary
Content-Disposition: form-data; name="signature"
Content-Type: application/json
{
  "signature": "84e0982a082024fac464bc5eb69c0745",
}
--formdataboundary
Content-Disposition: form-data; name="ObjectRecognitionModel";
Content-Type: application/octet-stream
[4CD563172A936520162652AA0CC6C9746D82020F7746865206A66F2726F7B96F7521]
--formdataboundary
```

Response:

```
HTTP/1.1 204 No Content
```

6.8.4.3 Write Configuration as Parameters

This method writes the configuration as parameter values to an entity.

Query Parameters:

This method does not support query parameters.

Request Headers:

```
Content-Type: application/json
```

Request Body:

Table 91 Request Body - Write configuration as parameters

Attribute	Type	Convention	Description
data	AnyValue	M	The value of the data. The type of the value is defined by the resource.
signature	string	O	Binary string with signature information (if required by the SOVD server / entity)

Examples:

- Writing configuration of an entity as a list of key-value pairs (APP settings)
- ECU config files

Response Status Codes:

Table 92 Response Status Codes - Write configuration as parameters

Status Code	Response Body	Description
204	--	Configuration written
400	DataError	The request body does either not contain all required parameter values or the signature for the provided values is wrong.
409	GenericError	Unable to write configuration due to unsatisfied precondition. E.g., after satisfying precondition “vehicle not in motion” this should be possible

Response Body:

This method does not provide a response body.

Example:

Request:

```
PUT {base_uri}/apps/AdvancedLaneKeeping/configurations/ALKConfig
Content-Type: application/json HTTP/1.1

{
  "data": {
    "MaximumSpeed": 100,
    "MinimumDistanceToLine": 10.0,
    "EmergencyLaneDistanceToLine": 5.0
  },
  "signature": "cz71a129fez26sw8"
}
```

Response:

```
HTTP/1.1 204 No Content
```

6.9 API Methods for Control of Operations

6.9.1 Introduction

An operation resource represents any executable diagnostic object, e.g., I/O controls, routines, and software functions. The SOVD API distinguishes between two different ways of executing an operation:

- Synchronous execution: The API method for executing the operation returns after the operation has been completed. The response includes the operation result.
- Asynchronous execution: The API method for executing the operation returns immediately and a client has to query the status of the execution which is provided as a temporary resource. In addition, the temporary resource can be used for updating the execution of the resource using a **`PUT`** method as long as the temporary resource exists. The temporary resource has to be removed explicitly using a **`DELETE`** method.

6.9.2 Ensuring Proximity Diagnostics

Certain operations should only be executed if their execution is supervised by a technician, e.g., to ensure that neither the vehicle nor a person takes damage. These operations require a proof of proximity which is performed by a challenge-response procedure. The implementation of the challenge-response mechanism is OEM-specific and not part of the SOVD specification.

There might be different levels of confirmation. E.g., for a remote control, the proximity confirmation only confirms that someone is within proximity. This does not necessarily have to be the one controlling the vehicle.

Examples for proximity confirmation could be:

- Request to press a specific button in the vehicle,
- Request to confirm on the screen in the vehicle,
- Scan a QR code displayed in the vehicle

6.9.3 Retrieve List of All Available Operations from an Entity

Method:

```
GET /{entity-path}/operations
```

Method Description:

This method returns all available operations defined for the given entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 93 Query Parameters - Retrieve list of all available operations from an entity

Parameter Name	Type	Convention	Description
`include-proximity-proof`	`boolean`	O	Specifies whether operations which require a proximity proof are returned or not. The default is `true`.
`include-schema`	`boolean`	O	Specifies whether the response should include schema information or not. The default is `false`.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 94 Response Status Codes - Retrieve list of all available operations from an entity

Status Code	Response Body	Description
200	Table 95	The request was successful.

Response Body:

Table 95 Response Body - Retrieve list of all available operations from an entity

Attribute	Type	Convention	Description
items	OperationDescription[]	M	The list of all operations of the entity.
schema	OpenAPI Schema	C	The schema definition of the response. Condition: Only provided if the query parameter include-schema is true.

Table 96 `OperationDescription` type

Attribute	Type	Convention	Description
id	string	M	Unique identifier for an operation
name	string	O	Operation name
translation_id	string	O	Identifier for translating the operations name
proximity_proof_required	boolean	M	If true, the execution of the operation requires a proof of proximity, and the offline capability description includes the attribute x-sovd-proximity-proof-required in the PathItem object. An SOVD client can use this information to filter operations in advance if a proximity proof cannot be provided.
asynchronous_execution	boolean	M	If true, the execution of the operation is asynchronous, and the offline capability description includes the attribute x-sovd-asynchronous-execution with the value true in the PathItem object.

Example:

Request:

```
GET {base_uri}/components/PowerSteering/operations HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "id": "SteeringAngleControl",
      "name": "Control the steering angle value",
      "translation_id": "tid1928653",
      "proximity_proof_required": false,
      "asynchronous_execution": true
    }
  ]
}
```

6.9.4 Get Details of a Single Operation

Method:

`GET /{entity-path}/operations/{operation-id}`

Method Description:

This method returns all the details on the specified operation of the given entity.

Path Parameters:

Table 97 Path Parameters - Get details of a single operation

Parameter Name	Type	Convention	Description
`operation-id`	`string`	M	Identifier for the operation

Query Parameters:

Table 98 Query Parameters - Get details of a single operation

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 99 Response Status Codes - Get details of single operation

Status Code	Response Body	Description
200	Table 100	The request was successful.

Response Body:

Table 100 Response Body - Get details of single operation

Attribute	Type	Convention	Description
item	OperationDescription	M	Description of the operation
proximity_challenge	ProximityChallenge	C	If proximity_proof_required is true, the challenge is provided. The challenge is used for proving proximity to the vehicle.
modes	Map of AnyValue[] (defined as additionalProperties see [25])	C	The modes in which the entity shall be for executing the operation. An SOVD client shall set the entity into one of the listed modes before executing the operation. If the attribute is not present, the SOVD server does not provide the information. If it is empty, the operation can be executed in any mode.
schema	PathItem Object	C	Complete description of the capabilities including the operation parameters of the operation as an OpenAPI specification. Condition: Only provided if the query parameter include-schema is true.

Table 101 `ProximityChallenge` type

Attribute	Type	Convention	Description
challenge	string	M	Challenge which has to be solved by the SOVD client to prove proximity (see 6.9.2).
valid_until	string:date-time	M	The time until which the challenge is valid

The potential capabilities of an operation are defined as:

- ***execute***: Starts the operation or adjusts an I/O control. The initial operation is executed using the **`POST`** method which creates an execution Identifier (see following subchapters). Subsequent adjustments are performed using a **`PUT`** method as the value should only be changed and no new execution should be created.
- ***stop***: Stops the operation or returns the control over the operation to the entity.
- ***freeze***: Freezes the current state of the executed I/O control (equivalent of UDS freeze current state).
- ***reset***: Requests that the I/O control is reset to its initial state.
- ***status***: Report the current status of the operation. The status includes if the operation is running or not, and additionally any values describing the current status of the operation in more detail (e.g., execution progress).

Table 102 shows how the mapping of capabilities to the I/O controls, ECU routines, and HPC operations is done.

Table 102 Operation capabilities relations to SOVD methods

Capability	I/O control	ECU routine	HPC operation	Method
execute	Short term adjustment	Start	Start	**`POST`** (see 6.9.5)
stop	Return control to ECU	Stop	Stop	**`DELETE`** (see 6.9.8)
execute	Short term adjustment	--	--	**`PUT`** (see 6.9.9)
freeze	Freeze current state	--	--	**`PUT`** (see 6.9.9)
reset	Reset to default	--	--	**`PUT`** (see 6.9.9)
status	Report current state	Request results	Status	**`GET`** (see 6.9.7)

Note: All capabilities except for the **`POST`** method are only available for operations which have the attribute `asynchronous_execution` set to `true`.

An OEM may define additional capabilities to support specific use cases.

The different capabilities are represented by using a `PathItem` object from the OpenAPI specification, which defines all the possible methods (**`GET`**, **`POST`**, **`PUT`**, and **`DELETE`**) which can be executed. If a capability is not supported, the corresponding method is not included.

As the **`PUT`** method is used to express multiple capabilities the attribute `x-sovd-capabilities` is introduced for an operation (OpenAPI `Operation` object). This attribute specifies as an array which capabilities are supported. The type of `x-sovd-capabilites` is an enumeration with the values in the column “Capability” from the table above.

Note: Unlike UDS, measurements which represent the current state (e.g., window position) of an I/O control are not returned as part of the I/O control as there is no standardized way of how an SOVD server can identify this. Thus, the measurements are included in the

data collection. An OEM specific extension can be used to associate the I/O control with the corresponding data resource.

Example:

Request:

```
GET {base_uri}/components/PowerSteering/operations/
SteeringAngleControl HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "item": {
    "id": "SteeringAngleControl",
    "name": "Control the steering angle value",
    "translation_id": "tid1928653",
    "proximity_proof_required": false,
    "asynchronous_execution": true
  },
  "modes": {
    "session": "EXTENDED"
  }
}
```

6.9.5 Start Execution of an Operation

Method:

`POST /{entity-path}/operations/{operation-id}/executions`

Method Description:

An operation may support multiple executions in parallel. If this is not supported, this method will return an error if an operation with the same `operation-id` is already being executed.

A successfully started operation is added to the executions collection if it is executed asynchronously. After the asynchronous execution has been finished, the execution resource remains existent and an SOVD client can receive the status. However, the status of the execution cannot be queried indefinitely as an SOVD server implementation may remove the resource, e.g., upon SOVD server restart.

Path Parameters:

Table 103 Path Parameter - Start execution of an operation

Parameter Name	Type	Convention	Description
`operation-id`	`string`	M	Identifier for the operation

Query Parameters:

This method does not support query parameters.

Request Headers:

`Content-Type: application/json`

Request Body:

Table 104 Request Body - Start execution of an operation

Attribute	Type	Convention	Description
`timeout`	`integer`	O	Timeout after which the SOVD server should terminate the execution of the operation by using the capability ***stop***. The time is specified in seconds. For extending the timeout a new control request for the operation has to be submitted by the SOVD client.
`parameters`	`AnyValue`	C	Parameters for executing the operation.
`proximity_response`	`string`	C	Response to proximity challenge if required by the operation.

Response Header:

The response for status code 202 includes the following header to redirect the SOVD client for requesting the status:
`Location:` {base_uri}`/{entity-path}/operations/{operation-id}/executions/{execution-id}`

Response Status Codes

When an operation is started (capability ***execute***) the status code indicates how the SOVD client should handle the operation.

Table 105 Response Status Codes - Start execution of an operation

Status Code	Response Body	Description
`200`	Table 106	The operation returned immediately, and the response body contains the final result of the operation (“synchronous execution”).
`202`	Table 107	The execution of the operation has been triggered by the SOVD server. The successful response code does not indicate that the operation has been started successfully. An SOVD client would then use the method for retrieving the status (see 6.9.7) to request the status of this particular execution instance of the operation (“asynchronous execution”).
`409`	GenericError	The method is currently not available, e.g., because the operation is still executing.

Note: The SOVD server or its underlying components might have a timeout implemented after which the actuation is terminated (e.g., for component protection purposes). If necessary, the behavior can be monitored by requesting the execution status.

Response Body:

Table 106 Response Body - Start execution of an operation - synchronous execution

Attribute	Type	Convention	Description
parameters	object	O	Response parameters of the operation depending on the capability.
error	GenericError	C	Condition: Set if an error occurred when the operation is executed.

Table 107 Response Body - Start execution of an operation - asynchronous execution

Attribute	Type	Convention	Description
id	string	M	Identifier for monitoring the execution of the operation and performing additional operations on the execution instance (e.g., ***freeze***).
status	ExecutionStatus	O	Status of the executed operation

Table 108 `ExecutionStatus` type

Value	Description
running	The execution of an operation is running.
completed	The execution of an operation has completed successfully.
failed	The execution of an operation has terminated with a fault.

Example:

Request:

```
POST
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/ex
ecutions HTTP/1.1
{
  "timeout": 120,
  "parameters": {
    "control-type": "absolute",
    "angle": 180.0
  }
}
```

Response:

```
HTTP/1.1 202 Accepted
Location:
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/
executions/fd34f39d-06e7-494b-af2d-8928e1458fb0
```

```
{
  "id": "fd34f39d-06e7-494b-af2d-8928e1458fb0",
  "status": "running"
}
```

Note: For a complete operation sequence example, please refer to A.4.2.

6.9.6 Get Executions of an Operation

Method:

```
GET /{entity-path}/operations/{operation-id}/executions
```

Method Description:

This method returns currently existing executions of the given operation.

Path Parameters:

Table 109 Path Parameters - Get executions of an operation

Parameter Name	Type	Convention	Description
operation-id	string	M	Identifier for the operation

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 110 Response Status Codes - Get executions of an operation

Status Code	Response Body	Description
200	Table 111	The request was successful.

Response Body:

Table 111 Response Body - Get executions of an operation

Attribute	Type	Convention	Description
items	string[]	M	List of identifiers of all execution instances of the operation

Example:

Request:

```
GET
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/ex
ecutions HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      "id": "fd34f39d-06e7-494b-af2d-8928e1458fb0"
    }
  ]
}
```

6.9.7 Get the Status of an Operation Execution

Method:

`GET /{entity-path}/operations/{operation-id}/executions/{execution-id}`

Method Description:

This method returns the current status of the operation execution.

Path Parameters:

Table 112 Path Parameters - Get the status of an operation execution

Parameter Name	Type	Convention	Description
`operation-id`	`string`	M	Identifier for the operation
`execution-id`	`string`	M	Identifier for monitoring the execution of the operation.

Query Parameters:

Table 113 Query Parameters - Get the status of an operation execution

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 114 Response Status Codes - Get the status of an operation execution

Status Code	Response Body	Description
`200`	Table 115	The request was successful.

Response Body:

Table 115 Response Body - Get the status of an operation execution

Attribute	Type	Convention	Description
`status`	ExecutionStatus	M	Status of the executed operation.
`capability`	Capability	M	Capability executed at the moment. This is either ***execute*** or the capability which has been used with a **`PUT`** request (see 6.9.5 and 6.9.9).
`parameters`	`AnyValue`	C	Response parameters of the operation depending on the capability. The type of the value is defined by the `schema`. Condition: If the capability provides response parameters.
`schema`	`OpenAPI Schema`	C	Schema of the operation executions. Condition: Only provided if the query parameter `include-schema` is `true`.
`error`	DataError`[]`	C	Array containing detailed descriptions about occurred errors during execution of the operation. Condition: Set if an error occurred when the operation is executed. The `path` attribute references erroneous parameters of the operation execution. If the `path` attribute is "/" then the operation itself failed and the error describes the reason for the failure.

Example:

Request:

```
GET
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/
executions/fd34f39d-06e7-494b-af2d-8928e1458fb0 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "status": "running",
  "capability": "execute",
  "parameters": {
    "control-type": "absolute",
    "angle": 120.3
  }
}
```

6.9.8 Stop the Execution of an Operation

Method:

`DELETE /{entity-path}/operations/{operation-id}/executions/{execution-id}`

Method Description:

This method terminates the execution of the operation and removes it. The status of the operation can no longer be requested.

Path Parameters:

Table 116 Path Parameters - Stop the execution of an operation

Parameter Name	Type	Convention	Description
`operation-id`	`string`	M	Identifier for the operation
`execution-id`	`string`	M	Identifier for monitoring the execution of the operation.

Query Parameters:

This method does not support query parameters.

Request Headers:

`Content-Type: application/json`

Request Body:

Table 117 Request Body - Stop the execution of an operation

Attribute	Type	Convention	Description
proximity_response	string	C	Response to proximity challenge if required by the operation.

Response Status Codes:

Table 118 Response Status Codes - Stop the execution of an operation

Status Code	Response Body	Description
204	--	The request was successful.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/
executions/fd34f39d-06e7-494b-af2d-8928e1458fb0 HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.9.9 Support for Execute / Freeze / Reset and OEM-Specific Capabilities

Method:

PUT /{entity-path}/operations/{operation-id}/executions/{execution-id}

Method Description:

This method executes the given capability on the provided operation execution.

Path Parameters:

Table 119 Path Parameters - Support for execute / freeze / reset and OEM-specific capabilities

Parameter Name	Type	Convention	Description
operation-id	string	M	Identifier for the operation
execution-id	string	M	Execution Identifier of the operation

Query Parameters:

This method does not support query parameters.

Request Headers:

```
Content-Type: application/json
```

Request Body:

Table 120 Request Body - Support for execute / freeze / reset and OEM-specific capabilities

Attribute	Type	Convention	Description
`capability`	Capability	M	Capability to be executed (one of ***execute***, ***freeze***, or ***reset*** as well as the OEM specific capabilities.
`timeout`	`integer`	O	Timeout after which the SOVD server should terminate the execution of the operation by using the capability ***stop***. The time is specified in seconds. For extending the timeout a new control request for the operation has to be submitted by the SOVD client.
`parameters`	`AnyValue`	C	Parameters for executing the operation.
`proximity_response`	`string`	C	Response to proximity challenge if required by the operation.

Response Header:

The response for status code 202 includes the `Location` header to redirect the SOVD client for requesting the status:

`Location:` {base_uri}`/{entity-path}/operations/{operation-id}/executions/{execution-id}`

Response Status Codes:

Table 121 Response Status Codes - Support for execute / freeze / reset and OEM-specific capabilities

Status Code	Response Body	Description
`202`	Table 107	The execution of the capability has been triggered by the SOVD server. The successful response code does not indicate that the operation has executed the capability successfully. An SOVD client would then use the method for retrieving the status (see 6.9.7) to request the

		status of this particular execution instance of the operation.

Response Body:

See Table 107.

Example:

Request:

```
PUT
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/
executions/fd34f39d-06e7-494b-af2d-8928e1458fb0 HTTP/1.1
{
  "capability": "reset",
  "timeout": 60
}
```

Response:

```
HTTP/1.1 202 Accepted
Location:
{base_uri}/components/PowerSteering/operations/SteeringAngleControl/
executions/fd34f39d-06e7-494b-af2d-8928e1458fb0

{
  "id": "fd34f39d-06e7-494b-af2d-8928e1458fb0",
  "status": "running"
}
```

6.10 API Methods for Support of Target Modes

6.10.1 Introduction

As a prerequisite for specific interactions with an entity, e.g., the execution of operations or changing data/configurations of an entity, the entity may have to be in a specific state in order to successfully process related requests. To enable SOVD clients to explicitly control and prepare such states of target entities (i.e., ***Component***s, ***App***s), the SOVD API provides information about current and possible states of target entities and supports any triggering required for state transitions. The state is maintained on the vehicle side. A concrete example in the CDA context is setting an ECU to a specific diagnostic session as a prerequisite to execute its operations. By explicitly setting the state, the SOVD client can control stateful hardware which is required by scenarios where a mode cannot be determined automatically, or performance is crucial. In the following, controllable states of target entities are called “modes”. A common basis is specified on how such modes can be served and controlled through an SOVD server via the SOVD API.
In an HPC context modes might be useful, e.g., for controlling that an input receives its value from a different source such as a simulation environment.
OEMs can use this as a basis to expose and interact with their introduced modes via the SOVD API.
The SOVD API supports two different ways of specifying which mode is required for the communication with an entity:

- Explicit mode control: The SOVD client defines which modes shall be activated prior to the API method call by the entity.
- Hint-based mode control: The SOVD client includes the modes which should be activated by the entity in the API method call.

6.10.2 Retrieve List of All Supported Modes of an Entity

Method:

```
GET /{entity-path}/modes
```

Method Description:

This method returns all available modes defined for the given entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 122 Query Parameters - Retrieve list of all supported modes of an entity

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 123 Response Status Codes - Retrieve list of all supported modes of an entity

Status Code	Response Body	Description
`200`	Table 124	The request was successful.

Response Body:

Table 124 Response Body - Retrieve list of all supported modes of an entity

Attribute	Type	Convention	Description
`items`	ModeCollectionItem`[]`	M	The list of available modes of an entity.
`schema`	`OpenAPI Schema`	C	The schema definition of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 125 `ModeCollectionItem` type

Attribute	Type	Convention	Description
`id`	`string`	M	The resource identifier of a mode of an entity.
`name`	`string`	O	The name of a mode of an entity.
`translation_id`	`string`	O	Translation identifier for the name

Example:

Request:

```
GET {base_uri}/components/PowerSteering/modes HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "id": "session",
      "name": "Diagnostic session"
    }
  ]
}
```

6.10.3 Get Details of a Single Mode of an Entity

Method:

`GET /{entity-path}/modes/{mode-id}`

Method Description:

This method returns all the details on the specified mode of the given entity.

Path Parameters:

Table 126 Path Parameters - Get details of a single mode of an entity

Parameter Name	Type	Convention	Description
`mode-id`	`string`	M	The identifier of a mode defined for an entity.

Query Parameters:

Table 127 Query Parameters - Get details of a single mode of an entity

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 128 Response Status Codes - Get details of a single mode of an entity

Status Code	Response Body	Description
`200`	Table 129	The request was successful.

Response Body:

Table 129 Response Body - Get details of a single mode of an entity

Attribute	Type	Convention	Description
`name`	`string`	O	The name of the mode.
`translation_id`	`string`	O	The identifier for translating the `name`.
`value`	`AnyValue`	M	The value of the mode.
`schema`	`OpenAPI Schema`	O	The schema of the mode resource. The default value for the mode is defined by the attribute `default`. Condition: Only provided if the query parameter `include-schema` is `true`.

Example:

Request:

```
GET {base_uri}/components/PowerSteering/modes/session?include-schema=true HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "name": "Diagnostic session",
```

```
    "value": "DEFAULT",
    "schema": {
      "type": "object",
      "required": [
        "id",
        "name",
        "value"
      ],
      "properties": {
        "id": {
          "type": "string"
        },
        "name": {
          "type": "string"
        },
        "value": {
          "type": "string",
          "enum": [
            "DEFAULT",
            "EXTENDED",
            "PROGRAMMING"
          ],
          "default": "DEFAULT"
        }
      }
    }
  }
```


6.10.4 Explicit Control of Entity States via Their Defined Modes

Method:

PUT /{entity-path}/modes/{mode-id}

Method Description:

The method allows to control the state of the specified mode of the given entity explicitly.

Path Parameters:

Table 130 Path Parameters - Explicit control of entity states via their defined modes

Parameter Name	Type	Convention	Description
mode-id	string	M	The resource identifier of the mode of an entity.

Query Parameters:

This method does not support query parameters.

Request Headers:

```
Content-Type: application/json
```

Request Body:

Table 131 Request Body - Explicit control of entity states via their defined modes

Attribute	Type	Convention	Description
value	AnyValue	M	The value to be set for the specified mode.
mode_ expiration	integer	M	Defines after how many seconds the mode expires and should therefore be automatically reset to the mode's default value. The capability description shall define the maximum expiration time and the default value of the mode, using the attribute `default`.

Response Status Codes

Table 132 Response Status Codes - Explicit control of entity states via their defined modes

Status Code	Response Body	Description
200	Table 133	Updated mode resource.
500	GenericError	The SOVD server received an error/negative response from the target entity when trying to set the mode to the provided value.

Response Body:

Table 133 Response Body - Explicit control of entity states via their defined modes

Attribute	Type	Convention	Description
id	string	M	The resource identifier of the mode.
value	AnyValue	M	The value of the mode.

Example:

Request:

```
PUT {base_uri}/components/PowerSteering/modes/session HTTP/1.1
{
  "value": "EXTENDED",
  "mode_expiration": 1200
}
```

Response:

```
HTTP/1.1 200 OK

{
  "id": "session",
  "value": "EXTENDED"
}
```

6.10.5 Hint-Based Control of Entity States via Their Defined Modes

An SOVD client may provide a hint to the SOVD server by providing a mode with any request as a request header:

```
X-SOVD-Mode: {mode-id};{mode-value}
```

Multiple modes are set in the header by using a comma. While processing the request the SOVD server shall set the entity to the mode given in the header.

6.11 API Methods for Locking

6.11.1 Introduction

The execution of some operations of an entity may require a lock so that the execution of this operation does not interfere with the execution of other operations, or the same operation performed by a different SOVD client. For example, the execution of an I/O control of a classic ECU would interfere with the execution of the same I/O control by a different SOVD client.

The standard provides the support for locking an entity by an SOVD client. The SOVD client then has exclusive access to all resources of the entity which require a lock. Only one lock per entity is allowed.

If one entity is locked, the sub entities below are also in the scope of this lock. It is not possible to lock those sub-entities individually in this case. It is also not possible to lock the entities above in this case. This means that there can be only one lock along the path to the entity.

The online and offline capability description describe that a lock is required for accessing a particular resource by introducing the attribute `x-sovd-lock-required`. This attribute is defined in the scope of an Operation or Path Item Object of the OpenAPI specification:

```
paths:
  /components/PowerSteering/operations/SteeringAngleControl/executions
    post:
       operationId: "SteeringAngleControl"
       x-sovd-lock-required: true
```

The attribute is specified for the Path Item if any action (HTTP method) on the resource requires a lock. If the attribute is specified for a specific action (e. g. **`POST`**) on the resource, then the lock is only required for performing this particular method. For example, for the execution of an operation the temporary resource **`/{entity-path}/operations/{operation-id}/executions/{execution-id}`** is used. For a given execution of an operation, querying the status with the **`GET`** method may not require a lock but changing the execution with a **`PUT`** method could require a lock.

Table 134 `x-sovd-lock-required` attribute

Parameter Name	Type	Convention	Description
`x-sovd-lock-required`	`boolean`	O	If `true` performing an action on the resource or a particular HTTP method requires a lock of the entity. By default, the attribute has the value `false`.

The SOVD server may associate the acquired lock with the authorization token of an SOVD client in order to identify the ownership of the lock.

All resources which require a lock can only be accessed by an SOVD client if this SOVD client has acquired a lock before. If an SOVD client tries to access a resource which requires a lock without having locked the entity previously, the HTTP status code 409 (Conflicted) will be sent as a response. This status code is also used when an SOVD client tries to acquire a lock and the request cannot be fulfilled because the lock is already acquired by a different SOVD client. Resources which can be accessed without requiring a lock shall still be accessible by SOVD clients when a lock is set for an entity. For example, if the `faults` resource collection defines that the **`GET`** method can be accessed without a lock, it can still be accessed by another client when the whole entity is locked.

The SOVD Client defines an expiration time for each lock it acquires and is able to extend that time if required. However, the operation – e. g. an I/O control execution or a software update – for which a lock has been acquired may run longer than the time for which the lock has been acquired. Once the lock times out, the SOVD server will terminate all resources – for example an operation – which is associated with the expired lock. This implies, that the lock resource as well as temporary resources – for example an execution resource – are deleted and further access to them by an SOVD client results in 404 (not found) status code. An SOVD client can only acquire the lock after the associated resources have been terminated.

6.11.2 Acquire a Lock on an Entity

Method:

```
POST /{entity-path}/locks
```

Method description:

This method acquires a lock from the SOVD server for the given entity to reserve its resources for exclusive access of a single SOVD client, i.e., prevent other SOVD clients from operating on the resources in parallel. Any further requests against the SOVD API towards controlling an operation (actuators, routines, and input control), e.g., start the execution of an operation, is rejected by the SOVD server if the required lock is not acquired by the same SOVD client.
Once the lock is not required anymore, the SOVD client can delete the created lock resource by a corresponding **`DELETE`** request.
In addition, an SOVD client shall provide a relative expiration time after which the SOVD server should release the lock automatically. As a result, the SOVD server deletes the corresponding lock resource. The SOVD server might reject the requested expiration time if it is too long.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Body:

Table 135 Request Body - Acquire a lock on an entity

Attribute	Type	Convention	Description
`lock_expiration`	`integer`	M	Defines after how many seconds the lock expires and should therefore be automatically released by the SOVD server. The expiration time starts when the request is processed by the server. The capability description shall define the maximum expiration time.

Response Header:

The `Location` header redirects the SOVD client for requesting the status:
`Location: {base_uri}/{entity-path}/locks/{lock-id}`

Response Status Codes

Table 136 Response Status Codes - Acquire a lock on an entity

Status Code	Response Body	Description
`201`	Table 137	The SOVD server has successfully created a new lock resource and acquired a lock on all relevant resources.
`409`	GenericError	The SOVD server rejected the creation of the lock resource.

Response Body:

Table 137 Response Body - Acquire a lock on an entity

Attribute	Type	Convention	Description
`id`	`string`	M	The identifier of the created lock resource.

Example:

Request:

```
POST {base_uri}/components/PowerSteering/locks HTTP/1.1
```

```
{
  "lock_expiration": 3600
}
```

Response:

```
HTTP/1.1 201 Created
Location: {base_uri}/components/PowerSteering/locks/15c3e824-8278-44e1-
b0b2-
3bc16e4522f2

{
  "id": "15c3e824-8278-44e1-b0b2-3bc16e4522f2"
}
```

6.11.3 Get All Acquired Locks of an Entity

Method:

GET /{entity-path}/locks

Method description:

The method returns the acquired locks for the given entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Response Headers:

The method has no response headers.

Response Status Codes:

Table 138 Response Status Codes - Get all acquired locks of an entity

Status Code	Response Body	Description
200	Table 139	The request was successful.

Response Body:

Table 139 Response Body - Get all acquired locks of an entity

Attribute	Type	Convention	Description
items	LockInfo[]	M	Collection of Locks acquired for the entity. **Note:** In this version of the standard only one lock is

			possible for the whole entity.

Table 140 `LockInfo` type

Attribute	Type	Convention	Description
`id`	`string`	M	Identifier of the lock.
`owned`	`boolean`	M	If `true`, the SOVD client which performed the request owns the lock. The value is always `false` if the entity is not locked.

Example:

Request:

```
GET {base_uri}/components/PowerSteering/locks HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items":[
    {
      "id": "15c3e824-8278-44e1-b0b2-3bc16e4522f2",
      "owned": true
    }
  ]
}
```

6.11.4 Get a Single Active Lock of an Entity

Method:

`GET /{entity-path}/locks/{lock-id}`

Method description:

The method returns all the details on the specified lock of the given entity. For extending the expiration of a lock the method described in 6.11.5 has to be invoked.

Path Parameters:

Table 141 Path Parameters - Get a single active lock of an entity

Parameter Name	Type	Convention	Description
`lock-id`	`string`	M	The resource identifier of the lock resource to be requested.

Query Parameters:

This method does not support query parameters.

Response Status Codes:

Table 142 Response Status Codes - Get a single active lock of an entity

Status Code	Response Body	Description
200	Table 143	The request was successful.

Response Body:

Table 143 Response Body - Get a single active lock of an entity

Attribute	Type	Convention	Description
lock_ expiration	string:date -time	M	A date-time string ([21], [19]), that defines at which point in time the lock expires and would therefore be automatically released by the SOVD server. The SOVD server always returns an absolute date and time in UTC based on the relative time provided by the SOVD client upon lock creation.

Example:

Request:

```
GET {base_uri}/components/PowerSteering/locks/15c3e824-8278-44e1-b0b2-3bc16e4522f2 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "lock_expiration": "2021-06-22T11:47:22Z"
}
```

6.11.5 Modify the Expiration Time of an Acquired Lock on an Entity

Method:

PUT /{entity-path}/locks/{lock-id}

Method description:

This method modifies an acquired lock by providing a new expiration time. The modification is only valid for the SOVD client that has already acquired the corresponding lock.

Path Parameters:

Table 144 Path Parameters - Modify the expiration time of an acquired lock on an entity

Parameter Name	Type	Convention	Description
lock-id	string	M	Identifier of a previous lock owned by the SOVD client.

Query Parameters:

This method does not support query parameters.

Request Headers:

The method has no special request headers.

Request Body:

Table 145 Request Body - Modify the expiration time of an acquired lock on an entity

Attribute	Type	Convention	Description
lock_ expiration	integer	M	Defines after how many seconds the lock expires and should therefore be automatically released by the SOVD server. The expiration time starts when the request is processed by the server and is not added to the original expiration time.

Response Status Codes

Table 146 Response Status Codes - Modify the expiration time of an acquired lock on an entity

Status Code	Response Body	Description
204	--	The request was successful.
409	GenericError	The SOVD server rejected the extension of the lock resource.

Response Body:

The method has no response body.

Example:

Request:

```
PUT {base_uri}/components/PowerSteering/locks/15c3e824-8278-44e1-b0b2-3bc16e4522f2 HTTP/1.1
Content-Type: application/json
```

```
{
  "lock_expiration": 1800
}
```

Response:

```
HTTP/1.1 204 No Content
```

6.11.6 Release an Acquired Lock on an Entity

Method:

`DELETE /{entity-path}/locks/{lock-id}`

Method description:

This method releases an acquired lock by deleting the created lock resource.

Request Headers:

The method has no special request headers.

Path Parameters:

Table 147 Path Parameters - Release an acquired lock on an entity

Parameter Name	Type	Convention	Description
lock-id	string	M	The resource identifier of the lock resource to be deleted.

Query Parameters:

This method does not support query parameters.

Response Status Codes

Table 148 Response Status Codes - Release an acquired lock on an entity

Status Code	Response Body	Description
204	--	The request was successful.
409	GenericError	The SOVD server rejected the deletion of the lock resource for some reason.

Response Body:

The method has no response body.

Example:

Request:

```
DELETE {base_uri}/components/PowerSteering/locks/15c3e824-8278-44e1-
b0b2-3bc16e4522f2 HTTP/1.1
```

Response:
```
HTTP/1.1 204 No Content
```

6.12 API Methods for Software Update

6.12.1 Introduction

In the classic approaches the diagnostic tester (as an SOVD client outside of the vehicle) has control over the flash process and the vehicle simply executes the prescribed steps. In contrast to this approach, the SOVD API puts the control into the vehicle and leverages the OTA capabilities of a connected vehicle to retrieve and install the available software updates.

The SOVD API distinguishes between the two update procedures

- stepwise software update and
- automated software update

to accommodate the different software update scenarios. With the stepwise software update the SOVD client can control which update is installed and when the update is prepared and executed. The automated software update puts the SOVD server in control of the complete update process for the update package selected by the client.

The possible update packages are provided by the SOVD API described in 6.12.2. An update package is always self-contained, i.e., if an update requires the update of multiple ECUs (e.g., engine and gear shift) these updates are included in the update package and are installed in one atomic step. If an update package includes multiple entities, these entities shall be located below the given entity path. E.g., if different top-level ***Component***s of the vehicle are updated, the common entity path will be the SOVD server entity. As the SOVD API does not restrict which actions can be performed as part of a software update, the update routine may also include parameterization and other activities. An additional consequence of an atomic update is that an update is rolled back completely in case of an error.

Although the SOVD API only defines the resource `/updates` on an SOVD server, which is used for updating all entities under the control of that SOVD server, an entity may provide this resource as well. If an entity provides the `updates` resource, specific use cases like updating individual entities in engineering or production can be supported.

6.12.1.1 Update Origins

The retrieval of update packages supports the definition of an origin to specify in which environment or by whom the software update is triggered by the SOVD client.

Table 149 `UpdateOrigins` type

Value	Description
`remote`	The update is performed over-the-air and no technician intervention is possible. Based on the requested version, authorization, network connection, etc. the SOVD server (or OEM backend) defines which updates are applicable.
`proximity`	The update is performed in a workshop and potentially technician intervention is possible.

An OEM may define additional origins. These origins shall be prefixed with **`x-<oem>`** and can contain alphanumeric characters, hyphens, and underscores.

6.12.1.2 Stepwise Software Update

The software update is divided into the following steps:

- Step 1: Get applicable updates (see 6.12.2):
  The vehicle provides the applicable updates to the SOVD client. The SOVD client then selects which update should be installed. The applicable updates may either by queried from an OTA backend when the command is executed, or the vehicle provides a list of updates which have been pushed to the vehicle by the OTA backend before.
- Step 2: Prepare update (see 6.12.5):
  The update selected by the SOVD client is prepared for installation, by downloading it from the OTA backend, validating its applicability for the vehicle, installing it in B-memory, etc. After this step the update is ready to be installed.
- Step 3: Execute update *(see* 6.12.6*)*:
  The prepared update is installed and activated. If multiple ECUs are flashed as part of the update, an SOVD server may install these updates in parallel if possible. As part of this step also necessary configurations, setup procedures, … etc. are performed.

Step 1 may be executed independently. This step can be omitted, if the identifier for the update package – which is required in step 2 – can be obtained from a different source. However, steps 2 and 3 are closely related to each other as the preparation step defines which update is installed during the execution step.

6.12.1.3 Automated Software Update

The SOVD API provides a method for initiating the update and monitoring its execution status. An SOVD server is then responsible for ensuring the following aspects:

- Downloading the update package
- Installing and activating the update package

6.12.1.4 Autonomous Update Package

The SOVD API defines the special update package identifier `autonomous`. Autonomous update packages are used in scenarios where a selection of the update package by an SOVD client is not required, e.g., for installing the latest security fixes. This is also known as self-governed update.

The special update package identifier `autonomous` can only be used to get the concrete update package identifier for installing the update. The update package identifier `autonomous` cannot be used to install an update.

6.12.1.5 Example

This subchapter provides an overview of the idea behind the mechanisms. It is not intended to be a comprehensive list of the provided methods which are described in detail in the following subchapters.

As depicted in Figure 7, below the collection resource “/updates”, two different kinds of resources can exist: “autonomous” and update resources representing available update

packages, i.e., “update-1” and “update-4”. An update package can contain e.g., one or more updates for ECUs or other entities, files, containers, or any possible combination of them.

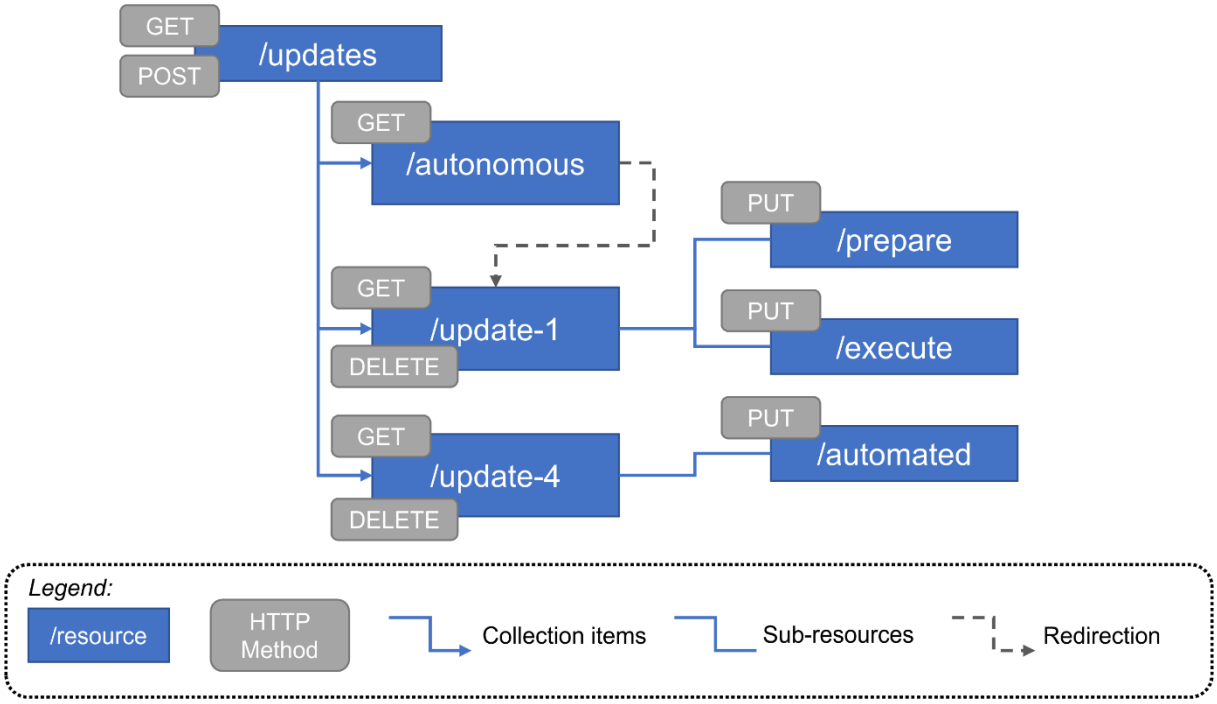

Figure 7: SOVD API sub-tree related to SW updates

As reflected by the HTTP methods attached to the resources shown in Figure 7, the usage of these resources can be as follows:

- **GET** {base_uri}**/updates** delivers a list containing “autonomous”, “update-1” and “update-4”.
- **GET** {base_uri}**/updates/autonomous** reports the update-id of an update package which is identified by the SOVD server to be an installable update at that point in time. With respect to the example shown in Figure 7, the SOVD server may return “update-1” as response.

The update package represented by the returned ID can then be installed either using:

- **PUT** {base_uri}**/updates/update-1/prepare**, followed by **PUT** {base_uri}**/updates/update-1/execute** (for the stepwise update) or by using
- **PUT** {base_uri}**/updates/update-4/automated**

With this structure, automated updates as well as stepwise updates can be handled using the same resource structure. If the SOVD server respectively the update package supports this, the SOVD client can even choose which method it wants to apply.

6.12.2 Retrieve List of All Updates

Method:

```
GET /updates
```

Method Description:

The method returns the available updates for the entity.

Path Parameters:

The method does not support path parameters.

Query Parameters:

Table 150 Query Parameters - Retrieve list of all updates provided by the entity

Parameter Name	Type	Convention	Description
`target-version`	`string`	O	Defines the version which is requested by the SOVD client. This version can be a specific version or a version pattern.
`origin`	UpdateOrigins	O	Defines from where the update is requested, e.g., inside a workshop or by an OTA update request. Default: remote

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 151 Response Status Codes - Retrieve list of all updates provided by the entity

Status Code	Response Body	Description
`200`	Table 152	The request was successful.

Response Body:

Table 152 Response Body - Retrieve list of all updates provided by the entity

Attribute	Type	Convention	Description
`items`	`string[]`	M	Update package identifier

Note: The SOVD standard predefines the special update package id `autonomous`, which is used for autonomous updates. This update package is included in the response if the autonomous update of the entity is supported.

Example (Remote):

Request:

```
GET {base_uri}/updates HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    "autonomous",
    "ADAS-v2.03.2154",
    "CAM-v1.18.8643"
  ]
}
```

Example (Workshop):

Request:

```
GET {base_uri}/updates?origin=proximity HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    "WC-v4.03.8413"
  ]
}
```

6.12.3 Get Details of Update

Method:

`GET /updates/{update-package-id}`

Method Description:

The method returns detailed information for a given update package.

Path Parameters:

Table 153 Path Parameters - Get details of update

Parameter Name	Type	Convention	Description
`update-package-id`	`string`	M	Identifier for the update package.

Query Parameters:

Table 154 Query Parameters - Get details of update

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 155 Response Status Codes - Get details of update

Status Code	Response Body	Description
200	Table 156	The request was successful.

Response Body:

Table 156 Response Body - Get details of update

Attribute	Type	Convention	Description
id	string	M	Identifier of the update package.
update_name	string	M	Name of the update
update_ translation_id	string	O	Translation identifier for the update_name.
automated	boolean	O	Specifies whether the update package can be installed automatedly or not. If missing, the attribute defaults to false.
origin	UpdateOrigins []	O	List of origins for which the update package is applicable. If the attribute is left out or empty, any origin is applicable.
notes	string	O	Release notes for the updates, may also be a URI.
notes_ translation_id	string	O	Translation identifier for the notes.
user_activity	string	O	Activities which have to be performed by the user after the update.
user_activity_ translation_id	string	O	Translation identifier for the user_activity.
preconditions	string	O	Preconditions which shall be fulfilled for installing the update (e.g., doors locked, parking, …). In general, not being able to fulfill the preconditions would block the preparation of the update.
preconditions_ translation_id	string	O	Translation identifier for the preconditions.
execution_	string	O	Description of the conditions which are required for executing

conditions			the update. May only be set if status is completed.
duration	integer	C	Time for which the vehicle will not be available when the update is installed and activated. The time is defined in seconds. If not set, the availability of the vehicle would not be affected.
size	integer	M	The download size of the update is defined in kilo bytes.
updated_components	string[]: uri-reference	O	A list of ***Component***s to be updated. The URIs are in the form of {entity-path}.
affected_components	string[]: uri-reference	O	The list documents which ***Component***s may have a different behavior etc. The focus lies on user-perceived functions and ***Component***s. The URIs are in the form of {entity-path}.
schema	OpenAPI Schema	C	Schema of the update package. Condition: Only provided if the query parameter include-schema is true.

Example:

Request:

```
GET {base_uri}/updates/autonomous HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "id":             "ADAS-v2.03.2154",
  "update_name":    "ADAS feature update to version 2.03.2154",
  "automated":      "true",
  "origins":        [ "remote", "proximity" ],
  "preconditions": "parking",
  "duration":       900,
  "size":           247000,
  "updated_components": [
    "{base_uri}/components/PowerSteering",
    "{base_uri}/apps/AdvancedLaneKeeping"
  ],
  "affected_components": [
    "{base_uri}/apps/AdvancedLaneKeeping"
  ]
}
```

6.12.4 Automated Installation of an Update

Method:

`PUT /updates/{update-package-id}/automated`

Method Description:

The method starts the automated installation of the update package. For this scenario, the SOVD server and the OEM's backend have full control over the update process and decide which action can be taken when.

The method returns immediately and redirects the SOVD client to a method for requesting the status and progress of the installation.

Path Parameters:

Table 157 Path Parameters - Automated installation of an update

Parameter Name	Type	Convention	Description
`update-package-id`	`string`	M	Identifier for the update package.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Header:

The response includes the following header to redirect the SOVD client for requesting the status:
`Location: {base_uri}/updates/{update-package-id}/status`

Response Status Codes:

Table 158 Response Status Codes - Automated installation of an update

Status Code	Response Body	Description
`202`	---	If no error is detected immediately (e.g., because the update package does not exist)
`409`	GenericError	An update is already in installation and not yet done or aborted. ErrorCode: `update-process-in-progress` `message: 'An update is already in preparation'`

		parameters: - key: id value: 'Identifier of update'
409	GenericError	Automated installation of the update package is not supported ErrorCode: update-automated-not-supported message: 'The package cannot be installed automatically'

Response Body:

This method does not provide a response body.

Example:

Request:

```
PUT {base_uri}/updates/ADAS-v2.03.2154/automated HTTP/1.1
```

Response:

```
HTTP/1.1 202 Accepted
Location: {base_uri}/updates/ADAS-v2.03.2154/status
```

6.12.5 Prepare Installation of an Update

Method:

PUT /updates/{update-package-id}/prepare

Method Description:

The method starts the preparation of the update package for installation. As part of this step the following tasks are performed for example:

- Validates if the update package is still installable (e.g., checking the software versions of the affected entities)
- Download of update package from OTA backend
- Validation of the integrity of update package
- Applying deltas
- Installing the update in B-memory

The method returns immediately and redirects the SOVD client to a method for requesting the status and progress of the preparation.

An SOVD server may remove prepared updates after an SOVD server specific amount of time. After a successful installation, the SOVD server will delete the update package automatically. If an update fails, it is in general not implicitly removed as the prepared package may be used for another attempt. An SOVD server may however remove the package if free space is required.

Path Parameters:

Table 159 Path Parameters - Prepare installation of an update

Parameter Name	Type	Convention	Description
`update-package-id`	`string`	M	Identifier for the update package.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Header:

The response includes the `Location` header to redirect the SOVD client for requesting the status:
`Location: {base_uri}/updates/{update-package-id}/status`

Response Status Codes:

Table 160 Response Status Codes - Prepare installation of an update

Status Code	Response Body	Description
`202`	---	If no error is detected immediately (e.g., because the update package identifier does not exist).
`409`	GenericError	An update is already in preparation and not yet done or aborted

Response Body:

This method does not provide a response body.

Note: The SOVD server may execute no operation at all, and the status of this step is "completed" as the OTA backend already prepared the update.

Example:

Request:
```
PUT {base_uri}/updates/WC-v4.03.8413/prepare HTTP/1.1
```

Response (Success):
```
HTTP/1.1 202 Accepted
Location: {base_uri}/updates/WC-v4.03.8413/status
```

Response (Error):

```
HTTP/1.1 409 Conflicted
{
  "error_code": "update-preparation-in-progress",
  "message": "An update is already in preparation",
  "parameters": {
    "id": "Identifier of update"
  }
}
```

6.12.6 Execute Installation of an Update

Method:

`PUT /updates/{update-package-id}/execute`

Method Description:

The method starts the installation and activation of the update package. As part of this step the following tasks are performed for example:

- Validation if the update package is still installable (e.g., checking the software versions of the updated entities)
- Installation of the update on all updated entities
- Configuration of the entities
- Execution of specific post install actions
- Activation of the update (e.g., by switching to B-memory)
- Rollback of update in case of an error

The method returns immediately and redirects the SOVD client to a method for requesting the status and progress of the installation.

After the update has been finished, the resource for the update package (and its sub-resources like status) remains existent and an SOVD client can receive the status of an update package. However, the status of the update package cannot be queried indefinitely as an SOVD server implementation may remove the update package resource, e.g., upon SOVD server restart or the retrieval of new update packages.

Path Parameters:

Table 161 Path Parameters - Execute installation of an update

Parameter Name	Type	Convention	Description
`update-package-id`	`string`	M	Identifier for the update package.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Header:

The response includes the `Location` header to redirect the SOVD client for requesting the status:
`Location: {base_uri}/updates/{update-package-id}/status`

Response Status Codes:

Table 162 Response Status Codes - Execute installation of an update

Status Code	Response Body	Description
202	---	If no error is detected immediately (e.g., because the identifier does not exist).
409	GenericError	Another update is already executed. ErrorCode: `update-execution-in-progess` `message: 'An update is already executed'`

Response Body:

This method does not provide a response body.

Example:

Request:
```
PUT {base_uri}/updates/WC-v4.03.8413/execute HTTP/1.1
```
Response:
```
HTTP/1.1 202 Accepted
Location: {base_uri}/updates/WC-v4.03.8413/status
```

6.12.7 Get Status of an Update

Method:

`GET /updates/{update-package-id}/status`

Method Description:

The method returns the status of the preparation or execution of the update.

Path Parameters:

Table 163 Path Parameters - Get status of an update

Parameter Name	Type	Convention	Description
update-package-id	string	M	Identifier for tracking the preparation or execution of the update.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 164 Response Status Codes - Get status of an update

Status Code	Response Body	Description
`200`	Table 165	The request was successful.

Response Body:

Table 165 Response Body - Get status of an update

Attribute	Type	Convention	Description
`phase`	Phase	M	Current phase of the update process.
`status`	Status	M	Status of the preparation or execution.
`progress`	`integer`	O	Progress of the preparation or execution in percent.
`subprogress`	Progress`[]`	O	Progress of the updated entities.
`step`	`string`	O	Description of the current step.
`step_translation_id`	`string`	O	Translation identifier for the step.
`error`	GenericError	O	Only set if status is `failed`.

Table 166 `Phase` type

Value	Description
`prepare`	Update preparation
`execute`	Update execution

Table 167 `Progress` type

Attribute	Type	Convention	Description
`entity`	`string: uri-reference`	M	URI of the entity.

status	Status	M	Status of the preparation or execution.
progress	integer	O	Progress of the preparation or execution in percent.
error	GenericError	O	Only set if status is failed.

Table 168 Status types

Status	Description
phase: prepare	
pending	The preparation of the update has not yet started.
inProgress	The preparation of the update has started.
failed	The preparation of the update has failed. Details on the reason are specified in the error attribute.
completed	The preparation of the update has been completed successfully, but the execution of the update has not yet been requested.
phase: execute	
pending	The preparation of the update has been completed, but the update itself has not yet been started.
inProgress	The installation of the update has started.
failed	The installation of the update has failed. Details on the reason are specified in the error attribute.
completed	The installation of the update has been completed successfully.

Note: If the preparation of an update fails, the request for the status would still use the response status code 200, as the status request was answered successfully. The attribute error, however, will contain information about the error that occurred. A response status code of 5xx would only be used if the status request could not be answered by the SOVD server.

Example:

Request:

```
GET {base_uri}/updates/ADAS-v2.03.2154/status HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "phase": "execute",
  "status": "inProgress",
  "progress": 87,
  "subprogress": [
    {
      "entity": "{base_uri}/components/PowerSteering",
      "status": "completed",
      "progress": 100
```

```
      },
      {
        "entity": "{base_uri}/apps/AdvancedLaneKeeping",
        "status": "inProgress",
        "progress": 75
      }
    ]
  }
```

6.12.8 Delete Update Package from an SOVD Server (Optional)

Method:

`DELETE /updates/{update-package-id}`

Method Description:

The method deletes an update package, which is not in the `phase execute`. The resource is afterwards not available anymore.

Path Parameters:

Table 169 Path Parameters - Delete update package from an SOVD server

Parameter Name	Type	Convention	Description
`update-package-id`	`string`	M	Identifier for the update package created by the SOVD server.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 170 Response Status Codes - Delete update package from an SOVD server

Status Code	Response Body	Description
`204`	`--`	The request was successful.
`405`	`--`	Deleting is not foreseen for an autonomous update-package-id or update-package is in `phase` execute.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE {base_uri}/updates/WC-v4.03.8413 HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.12.9 Register an Update at the SOVD Server

Method:

`POST /updates`

Method Description:

The method registers a new update at the SOVD server which can be installed afterwards. After creating the update, the SOVD client has to use the methods described before for installing the update (automated or prepare & execute).

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

`Content-Type`: OEM-specific

Request Body:

The content of the request body is OEM and SOVD server implementation specific as it provides the necessary information to the SOVD server to create a new update package including the required files.

Response Header:

The response includes the `Location` header to redirect the SOVD client for interacting with the package:
`Location: {base_uri}/updates/{update-package-id}`

Response Status Codes:

Table 171 Response Status Codes - Register an update at the SOVD server

Status Code	Response Body	Description
201	Table 172	The request was successful.

Response Body:

Table 172 Response Body - Register an update at the SOVD server

Attribute	Type	Convention	Description
id	string	M	Identifier for the update package created by the SOVD server.

Example:

Request:

```
POST {base_uri}/updates HTTP/1.1
Content-Type: application/json

{
  "id":         "SW-v2.1.9525",
  "source":     "USB",
  "filename":   "UpdatePackage-V2.1.9525.pkg",
  "signature": "d9df94cf1f5f5413fa325f8b52bd28a4"
}
```

Response:

```
HTTP/1.1 201 Created
Location: {base_uri}/updates/SW-v2.1.9525

{
  "id": "SW-v2.1.9525"
}
```

6.13 API Methods for Handling of Bulk Data

6.13.1 Introduction

The methods defined in this subchapter enable an SOVD client to manage bulk data such as kernel-dump files, temporarily generated log-files, etc. An SOVD client can download, and upload bulk data including proof-of-origins (security artefacts) and delete the data. The categories of the bulk data are defined by an OEM-specific BulkDataCategory type.

For transferring bulk data standard HTTP mechanism such as multipart request and response bodies are used as defined by RFC 1341 [27] and RFC 1847 [15]. There might be use cases, in which due to technical limitations a special API shall be used, e.g., to prepare and transfer the bulk data using a proprietary cloud data provider. How this is realized is not part of the SOVD API.

One example may be that log files increase over time, especially with higher log-levels activated. It might happen that a specific bulk data resource exceeds the limits of a single response. As a consequence, a resource that was accessible earlier may no longer be accessible by the same SOVD client with the same access rights. In this case, the SOVD API should still allow an SOVD client with the proper access rights to retrieve this bulk data resource.

6.13.1.1 Resource Organization

Figure 8 gives an overview how the bulk data resources of an entity are organized. Similar to data, the BulkDataCategory type is used for logical grouping of bulk data types. For example, “maps” and “pois” are different categories of bulk data which are typically handled differently by the navigation unit.

Using the methods described here a user can access the meta information of a bulk data category, can upload, download, and delete the corresponding resource. The access to the resources is based on the SOVD client’s authorization.

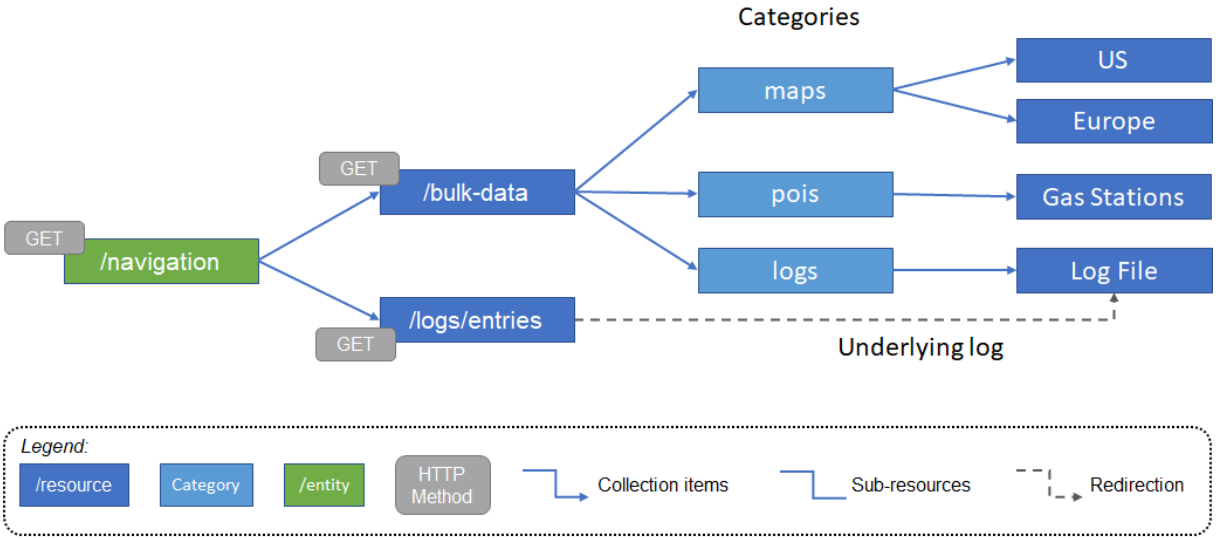

Figure 8 Resource organization based on categories

In addition, Figure 8 also shows the relationship between the standard resource logs and the log file from which the log resource retrieved the log entries. The underlying log file is accessible as a bulk data resource if required.

6.13.1.2 Encoding of Bulk Data

The bulk data transferred between the SOVD server and SOVD client is based on MIME (RFC 1341) [27] and MIME security (RFC 1847) [15] as a mandatory `Content-Type` header of the request or response. With this header the content of the body is defined and the SOVD server and SOVD client know how the content has to be interpreted.

Bulk data typically uses the `Content-Type application/octet-stream`. However, an SOVD client or SOVD server may always use a more specific MIME type than `application/octet-stream` if applicable. As an SOVD client may require additional information – e.g., the name of a file – additional information is provided as a JSON object. Thus, the typical body uses the `Content-Type multipart/form-data`.

For signing bulk data, the mechanisms defined by RFC 1847 [15] can be used, e.g., the `Content-Type multipart/signed` is used to store the signature for the bulk data together with the bulk data itself.

6.13.2 Retrieve List of all Bulk Data Categories

Method:

```
GET /{entity-path}/bulk-data/
```

Method Description:

This method provides the list of bulk data categories supported by an entity. The result contains all available and based on the authorization accesible bulk data categories. Certain bulk data categories are hidden based on the user's authorization.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 173 Query Parameters - Retrieve list of all bulk data categories

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 174 Response Status Codes - Retrieve list of all bulk data categories

Status Code	Response Body	Description
`200`	Table 175	The request was successful.

Response Body:

Table 175 Response Body - Retrieve list of all bulk data categories

Attribute	Type	Convention	Description
`items`	BulkDataCategory `[]`	M	The bulk data categories supported by the entity.
`schema`	`OpenAPI Schema`	C	The schema definition of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 176 Examples for `BulkDataCategory` type

Value	Description
`maps`	OEM specific category 1
`pois`	OEM specific category 2
`logs`	OEM specific category N

Note: These enumeration values are an example and not normative.

Example:

Request:

```
GET {base_uri}/apps/Navi/bulk-data HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    "maps",
    "pois",
    "logs"
  ]
}
```

6.13.3 Read Bulk Data Meta Data

Method:

`GET /{entity-path}/bulk-data/{category}`

Method Description:

This method provides the list of BulkDataDescriptors for a specific category available and accessible for an SOVD client. The possible categories can be retrieved either using the interactive capability description method Retrieve List of all Bulk Data Categories or the offline capability description.

Note: Filtering can be done with proprietary mechanisms on the SOVD server-side or on the SOVD client-side based on the provided descriptors.

Path Parameters:

Table 177 Path Parameters - Read bulk data meta data

Attribute	Type	Convention	Description
`category`	BulkData Category	M	The category represents a logical grouping of bulk data exposed by the SOVD server implementation, e.g., POIs stored on a navigation unit.

Query Parameters:

Table 178 Query Parameters - Read bulk data meta data

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

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 179 Response Status Codes - Read bulk data meta data

Status Code	Response Body	Description
`200`	Table 180	The request was successful.

Response Body:

Table 180 Response Body - Read bulk data meta data

Attribute	Type	Convention	Description
`items`	BulkDataDescriptor`[]`	M	List of BulkDataDescriptors
`schema`	`OpenAPI Schema`	O	The schema of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 181 `BulkDataDescriptor` type

Attribute	Type	Convention	Description
`id`	`string`	M	A unique identifier for the bulk data managed by the SOVD server to identify the bulk data.
`mimetype`	`string`	M	Specifies the MIME type of the bulk data.
`name`	`string`	C	Name / description of the bulk data.
`translation_id`	`string`	O	Identifier for translating the name.
`size`	`integer`	C	Size of the bulk data in bytes.
`creation_date`	`string: date-time`	C	Time and date of the creation of the bulk data.

last_modified	string: date-time	C	Time and date of the last modification of the bulk data.
hash	string	C	Hash value of the bulk data for verifying the integrity of the bulk data after transfer.
hash_algorithm	string	C	Algorithm used to generate the hash.

- C: Only provided if the information is available for the bulk data.

Example:

Request:

```
GET {base_uri}/apps/Navi/bulk-data/maps HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "items": [
    {
      "id":       "EU",
      "name":     "OSM data for Europe",
      "mimetype": "application/vnd.osm+xml"
    },
    {
      "id":       "US",
      "name":     "OSM data for the United States of America",
      "mimetype": "application/vnd.osm+xml"
    }
  ]
}
```

6.13.4 Download and Upload Bulk Data

This subchapter defines the SOVD API methods for downloading and uploading bulk data from and to the SOVD server respectively. Transfer of bulk data, as described in this subchapter, is limited to a size that can be transferred within one shot by the underlying technology. For transfers that exceed the limit, other methods, such as an indirect synchronization with third-party services (like cloud storage providers) shall be used.

The security parameters to check the integrity (e.g., SHA256, MD5, etc.) of the bulk data or verification of proof-of-origin of the bulk data either through digital signatures (e.g., asymmetric keys) or any other method can also be optionally sent along with the bulk data during the transfer in a multi-part message.

6.13.4.1 Download Bulk Data

Method:

```
GET /{entity-path}/bulk-data/{category}/{bulk-data-id}
```

Method Description:

This method downloads the bulk data identified by `bulk-data-id` and `category` from the SOVD server. The SOVD server manages how `bulk-data-id` and `category` are mapped to the underlying data storage and how the retrieval of the bulk data based on the identifier is performed.
The method allows the transfer of a single bulk data.

Path Parameters:

Table 182 Path Parameters - Download bulk data

Parameter Name	Type	Convention	Description
`category`	BulkData Category	M	The category the requested bulk data belongs to.
`bulk-data-id`	`string`	M	Bulk data identifier

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not require any request headers. However, if an SOVD client requires a proof-of-origin, the `Accept` header has to be set with the value `multipart/signed`. The SOVD server will then sign the transferred data and set `Content-Type` header to `multipart/signed` instead of `multipart/form-data`. OEMs may, alternatively, decide to use `multipart/form-data` and include the security artefacts within the `application/json` part using OEM specific formats.

Response Status Codes:

Table 183 Response Status Codes - Download bulk data

Status Code	Response Body	Description
`200`	`OEM specific`	Provides content of the bulk data as part of the response.
`202`	`OEM specific`	The SOVD server accepted the request, but the data is not yet ready for retrieval. The SOVD client has to retry the request later on. An SOVD server may use the response header field `Retry-After` to inform the SOVD client after how many seconds a retry is reasonable.
`307`	`--`	Temporary Redirect The SOVD server does not provide access to the resource under the requested URI, but the SOVD client has to use a different URI for downloading the bulk data. The URI is provided using the `Location` response header.

Note: An SOVD server may technically not be able to provide a download for large bulk data or may have to do some processing before the download (e.g., compress the bulk data). By using the status codes 202 or 307, the SOVD server can separate the processing from the download step. For example, 202 is returned whilst the bulk data is uploaded to an external system like a cloud and 307 redirects the SOVD client to download the information from there.

Response Body:

The content of the response body is OEM specific. Depending on the implementation, the transferred bulk data can use `multipart/signed`, `multipart/form-data` with security parameters in a JSON object, or simply `application/octet-stream` (if it is not confidential information). The example below uses `multipart/form-data` with a JSON object for storing the signature of the bulk data.

Example:

Request:

```
GET {base_uri}/apps/Navi/bulk-data/pois/GasStations HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

Content-Type: multipart/form-data; boundary=formdataboundary
--formdataboundary
Content-Disposition: form-data; name="signature"
Content-Type: application/json
{
  "signature": "925b36e5bce4966f6ede24f0f4ec7a56",
}
--formdataboundary
Content-Disposition: form-data; name="GasStations";
filename="gas-stations.kml"
Content-Type: application/octet-stream
[File content]
--formdataboundary
```

6.13.4.2 Upload Bulk Data

Method:

`POST /{entity-path}/bulk-data/{category}`

Method Description:

This method uploads bulk data to the SOVD server and creates a resource for the bulk data. The request provides the content as well as any additional information, e.g., security artifacts, which are required to validate the content.

Path Parameters:

Table 184 Path Parameters - Upload bulk data

Parameter Name	Type	Convention	Description
`category`	BulkData Category	M	The category of the bulk data to be uploaded.

Query Parameters:

This method does not support query parameters.

Request Headers:

The following request headers for bulk data upload shall be used to specify the `Content-Type`, its size etc.

- `Content-Type`: Depends on Request Body
- `Content-Length`: Size of the content
- `Content-Disposition: form-data; name="<field>"; filename="<file>"`: Provides more information on the content and shall at least have the attribute `name` to give the SOVD server a hint on what is meant specifically. In addition, the attribute `filename` may be set to propose a file name in case the bulk data is stored on the file system. However, an SOVD server may also discard the proposed file name.

Request Body:

The content of the request body is OEM specific. Depending on the implementation, the transferred bulk data can use `multipart/signed`, `multipart/form-data` with security parameters in a JSON object, or simply `application/octet-stream` (if it is not confidential information).

Response Header:

The response includes the `Location` header to redirect the SOVD client to the created bulk data resource:
`Location: {base_uri}/{entity-path}/bulk-data/{category}/{bulk-data-id}`

Response Status Codes:

Table 185 Response Status Codes - Upload bulk data

Status Code	Response Body	Description
`201`	Table 186	The request was successful.

Response Body:

Table 186 Response Body - Upload bulk data

Attribute	Type	Convention	Description
id	string	M	Bulk data identifier created by the SOVD server to identify the bulk data.

Example:

Request:

```
POST {base_uri}/app/navigation/bulk-data/pois HTTP/1.1

Content-Length: 2740

Content-Type: multipart/form-data; boundary=formdataboundary
--formdataboundary
Content-Disposition: form-data; name="signature"
Content-Type: application/json
{
  "signature": "925b36e5bce4966f6ede24f0f4ec7a56",
}
--formdataboundary
Content-Disposition: form-data;name="GasStations";file-name="gas-
stations.kml"
Content-Type: application/octet-stream
[File content]
--formdataboundary
```

Response:

```
HTTP/1.1 Created 201
Location: {base_uri}/app/navigation/bulk-data/pois/GasStations

{
  "id": "GasStations"
}
```

6.13.5 Delete Bulk Data

There are two mechanisms to delete the bulk data present in a bulk data node

- Delete All Bulk Data Defined by Category
- Delete Specific Bulk Data Resource

6.13.5.1 Delete All Bulk Data Defined by Category

Method:

`DELETE /{entity-path}/bulk-data/{category}`

Method Description:

This method requests the deletion of all bulk data for the given category.

Path Parameters:

Table 187 Path Parameters - Delete all bulk data defined by category

Parameter Name	Type	Convention	Description
`category`	BulkData Category	M	The category of which all bulk data shall be deleted from.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 188 Response Status Codes - Delete all bulk data defined by category

Status Code	Response Body	Description
`200`	Table 189	The request was successful.
`409`	GenericError	Request cannot be completed due to a general conflict, e.g., none of the bulk data resource could be deleted.

Response Body:

Table 189 Response Body - Delete all bulk data defined by category

Attribute	Type	Convention	Description
`deleted_ids`	`string[]`	M	List of deleted bulk data identifiers belonging to the given category
`errors`	DeletionError`[]`	M	Bulk data identifiers which could not be deleted and why.

Table 190 `DeletionError` type

Attribute	Type	Convention	Description
`id`	`string`	M	Bulk data identifier, that could not be deleted.
`error`	GenericError	M	The reason why the bulk data resource could not be deleted.

Example:

Request:

```
DELETE {base_uri}/apps/Navi/bulk-data/maps HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

{
  "deleted_ids": [
    "US"
  ],
  "errors": [
    {
      "id": "EU",
      "error":
        {
          "error_code":  "vendor-specific",
          "vendor_code": "unable-to-delete",
          "message":     "Could not delete map currently in use."
        }
    }
  ]
}
```

6.13.5.2 Delete Specific Bulk Data Resource

Method:

`DELETE /{entity-path}/bulk-data/{category}/{bulk-data-id}`

Method Description:

This method requests the deletion of a specific bulk data resource of the given category.

Path Parameters:

Table 191 Path Parameters - Delete bulk data

Parameter Name	Type	Convention	Description
`category`	BulkData Category	M	The category of the bulk data to be deleted.
`bulk-data-id`	`string`	M	Bulk data identifier

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 192 Response Status Codes - Delete bulk data

Status Code	Response Body	Description
204	--	The request was successful.
409	GenericError	Request cannot be completed because the deletion of the bulk data resource failed, e.g., the bulk data resource cannot be deleted as it is still in use.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE {base_uri}/apps/Navi/bulk-data/maps/US HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.14 API Methods for Logging

6.14.1 Introduction

The SOVD logging methods provide access to aggregated log information. Critical messages should always be included.

SOVD clients are able to specify by log configurations what is aggregated from different contexts, e.g., ***Component***s or ***App***s.

Note: Tracing is explicitly excluded.

The log information provided via SOVD is not intended for evaluation in the workshop. It is intended to be used by software experts typically located in the OEMs back office. For workshop technicians, critical messages will be reported via the faults interface including appropriate additional information.

Note: The online capability description shall deliver both `logs/entries` and `logs/config` when querying the entity, if supported.

6.14.2 Retrieve List of All log Information

Method:

```
GET /{entity-path}/logs/entries
```

Method Description:

This method provides the log information aggregated by this entity.

Path Parameters:

This method does not support path parameters.

Query Parameters:

An SOVD client can optionally add query parameters to retrieve logging information only for a specific severity.

Table 193 Query Parameters - Retrieve list of all log information

Attribute	Type	Convention	Description
`severity`	Severity	O	SOVD specific enumeration defining log-levels
`include-schema`	`boolean`	O	Specifies whether the response should include schema information or not. The default is `false`.

Table 194 `Severity` type in relation to `ContextType`

Severity	ContextType	
	RFC5424 [36]	**AUTOSAR_DLT [6]**
`fatal`	Emergency Alert Critical	DLT_Fatal
`error`	Error	DLT_ERROR
`warn`	Warning	DLT_WARN
`info`	Notice Informational	DLT_INFO
`debug`	Debug	DLT_DEBUG DLT_VERBOSE

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 195 Response Status Codes - Retrieve list of all log information

Status Code	Response Body	Description
`200`	Table 196	The request was successful.

Response Body:

Table 196 Response Body - Retrieve list of all log information

Attribute	Type	Convention	Description
`items`	LogEntry`[]`	M	An array of log entries
`schema`	`OpenAPI Schema`	C	The schema definition of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 197 `LogEntry` type

Attribute	Type	Convention	Description
`timestamp`	`string:date-time`	M	Time and date when the log entry has been created.
`context`	Context	M	Definition of the context of the log message identifying the source of the entry.
`severity`	Severity	M	SOVD specific datatype defining log-levels
`msg`	`string`	M	The actual log message (text)
`href`	`string:uri-reference`	O	Reference to a URI for retrieving the underlying original log file in an application specific format

Table 198 `Context` attributes

Attribute	Type	Convention	Description
`type`	ContextType	M	Enumeration identifying the logformat used by the respective log source.
…			Depending on the value of the `type` attribute, further attributes will be present (see Table 199).

Table 199 ContextType dependent attributes

ContextType	Additional ContextType dependent Attributes	Type
RFC5424		
	host	string
	process	string
	pid	integer
AUTOSAR_DLT		
	session	string
	session_id	string
	application_id	string
	context_id	string
	message_id	string

Note: OEM-specific context types can be included using alpha-numeric characters, hyphens, and underscores and have to start with `x-`. An implementor of the SOVD API may introduce additional context types. These context types will not be supported by an off-the-shelf SOVD client.

Example:

Request:

```
GET {base_uri}/components/DrivingComputer/logs/entries HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "items": [
    {
      …
    },
    {
      "timestamp": "2021-07-20T00:00:04.387819Z",
      "context":
      {
        "type": "RFC5424",
        "host": "Linux",
        "process": "systemd",
        "pid": 1
      },
      "severity": "info",
      "msg": "Closed D-Bus User Message Bus Socket",
      "href": "{base_uri}/components/DrivingComputer/bulk-
       data/logs/server.log"
    },
    {
      …
    }
```

```
    ]
  }
```

6.14.3 Configure SOVD Logging

Method:

`PUT /{entity-path}/logs/config`

Method Description:

This method configures the log aggregation.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

Due to its dynamic nature, the context information is not provided by the online or offline capability description. The information has to be exchanged by other means.

Table 200 Request Body - Configure SOVD logging

Attribute	Type	Convention	Description
`items`	LogConfiguration[]	M	Array of LogConfiguration elements

Table 201 `LogConfiguration` type

Attribute	Type	Convention	Description
`context`	Context	M	Definition of the context of the log message. Parts not specified in the context are regarded as not set.
`severity`	Severity	M	The severity specifies the log-level for the given context.

Response Status Codes:

Table 202 Response Status Codes - Configure SOVD logging

Status Code	Response Body	Description
204	--	The configuration was accepted.
400	GenericError	Invalid configuration provided

Note: In general, changes to the Log configuration triggered via the SOVD client shall be persistent (i.e., still be available after a system restart to support enhanced startup logging). Implementations could deviate from this rule for specific reasons.

Response Body:

This method does not provide a response body.

Example:

Request:

```
PUT {base_uri}/components/DrivingComputer/logs/config HTTP/1.1
Content-Type: application/json

{
  "items": [
    {
      "context":
      {
        "type": "RFC5424",
        "host": "Linux",
        "process": "systemd"
      },
      "severity": "warn"
    }
  ]
}
```

Response:

```
HTTP/1.1 204 No Content
```

6.14.4 Retrieve the Current SOVD Logging Configuration

Method:

GET {entity-path}/logs/config

Method Description:

This method returns an array of configured SOVD LogConfiguration elements, specifying the different contexts for which a specific log configuration was created with the method Configure SOVD Logging.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Response Status Codes:

Table 203 Response Status Codes - Retrieve the current SOVD logging configuration

Status Code	Response Body	Description
200	Table 204	The request was successful.

Response Body:

Table 204 Response Body - Retrieve the current SOVD logging configuration

Attribute	Type	Convention	Description
contexts	LogConfiguration[]	M	An array of LogConfiguration elements

Example:

Request:

```
GET {base_uri}/components/DrivingComputer/logs/config HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
  "contexts": [
    {
      "context":
      {
        "type": "RFC5424",
        "host": "Linux",
        "process": "systemd",
        "pid": 1

      },
      "severity": "info"
    },
    {
      "context":
      {
        "type": "AUTOSAR_DLT",
        "session": "SES01",
        "session_id": "0x0001",
        "application_id": "WindowControl",
        "context_id": "STD0",
        "message_id": "msg000012"
      },
      "severity": "info"
```

```
        }
      ]
    }
```

6.14.5 Reset SOVD Logging Configuration to Default

Method:

`DELETE /{entity-path}/logs/config`

Method Description:

This method resets the current configuration to its default.

Path Parameters:

This method does not support path parameters.

Query Parameters:

This method does not support query parameters.

Request Headers:

This method does not use any specific header parameters.

Request Body:

No request body required.

Response Status Codes:

Table 205 Response Status Codes - Reset SOVD logging configuration to default

Status Code	Response Body	Description
204	--	Configuration has been reset successfully.
404	GenericError	The SOVD client tried to reset a configuration which does not exist.

Response Body:

This method does not provide a response body.

Example:

Request:

```
DELETE {base_uri}/components/DrivingComputer/logs/config HTTP/1.1
```

Response:

```
HTTP/1.1 204 No Content
```

6.15 Authentication of SOVD Clients (Informative)

6.15.1 Introduction

The goal of authentication is to enable an authorization and validation process to protect the resources of any accessible entity in a vehicle against misuses of features while accessing an SOVD server.

Another aspect of security might be end-to-end security, e.g., signatures for coding and programming data. This is part of the respective payload data and is not described in this SOVD API subchapter.

This subchapter defines requirements for the SOVD API. However, the process using the SOVD API features might differ depending on the security architecture as well as authorization protocols and is thus not standardized here. Only examples for the process are given.

A trust anchor needs to be established inside the vehicle. This trust anchor knows the SOVD servers and can give out tokens to a requestor to access those SOVD servers or their functionalities. Tokens may be implemented e.g., using JSON Web Tokens (see [11]).

The Vehicle AuthorizationServer provides the possibility to validate the issued tokens, which on the other hand enables the SOVD servers to decide how to deal with a request by the SOVD client.

The SOVD servers shall protect all resources against unauthorized access. The Vehicle AuthorizationServer does not directly prevent the SOVD server from being accessed by the SOVD client, but it does provide information, with what kind of access an SOVD client is authenticated. Only if the SOVD client has the appropriate access rights, the SOVD server shall grant the access to the requested resources.

Therefore, a receiving SOVD server may only process further requests with a valid token. If no token or an invalid token is provided with a request, the SOVD server needs to deny the request or redirect the SOVD client to the Vehicle AuthorizationServer. Therefore, a token provides the legitimacy to the SOVD client to communicate with the SOVD server.

The SOVD client should always possess a valid token from the `/token` URI before performing requests at an SOVD server. The AuthorizationServer creates a token after successfully validating the request.

The AuthorizationServer signs each token it creates with a private key so the token can be validated by the requested SOVD client.

With each request, the SOVD client shall send a valid token to the receiving SOVD server. Only if the token validation was successful, the SOVD server may execute the request.

SOVD offers basically two options related to authentication and authorization:

- **Online**: shown in Figure 9 (AuthorizationServer in backend and vehicle):
    a. AuthorizationServer in backend and vehicle established trust and are synchronized
    b. An SOVD client authenticates against the backend.
    c. The backend authenticates against the vehicle
    d. Backend issues the token to the SOVD client

- **Offline**: shown in Figure 10 (AuthorizationServer in the vehicle):
  a. An SOVD client can authenticate itself at the vehicle using its stored credentials (e.g., certificates)
  b. A token is negotiated between SOVD client and vehicle

Note: Both figures only illustrate the interaction between the components underlying to the SOVD security concept. For this purpose, only one HPC which hosts the SOVD API implementation, and the Vehicle AuthorizationServer are shown here. This might differ depending on the concrete vehicle architecture and deployment of SOVD servers across several HPCs.

In the following, the overall SOVD security concept based on these two options is introduced in more detail.

The concept is based on various components interacting together:

- A vehicle containing
  - One or multiple SOVD servers,
  - a classic mechatronic architecture with multiple ECUs,
  - an AuthorizationServer, e.g., contained in a security module
- A diagnostics SOVD client
- A Backend User Administration

6.15.1.1 Online Authentication and Authorization

An example for a typical sequence can be as shown in Figure 9:

1. The SOVD client authenticates at the Backend User Administration, e.g., via a log-in.
2. The SOVD client requests a token from the Backend AuthorizationServer. If allowed, the Backend AuthorizationServer issues a token to the SOVD client.
3. The Backend AuthorizationServer has a trusted communication with the Backend User Administration and is synchronized with the Vehicle AuthorizationServer(this communication is not scope of SOVD standardization). The Vehicle AuthorizationServer validates the SOVD client's authorization.
4. The SOVD client uses this token when issuing requests to the SOVD API.
5. The addressed SOVD server validates the token by asking the Vehicle AuthorizationServer for the validation key (public key fitting the private key which was used to create the signature for the token). This public key is then used to validate the signature with the public key and the access can be granted.

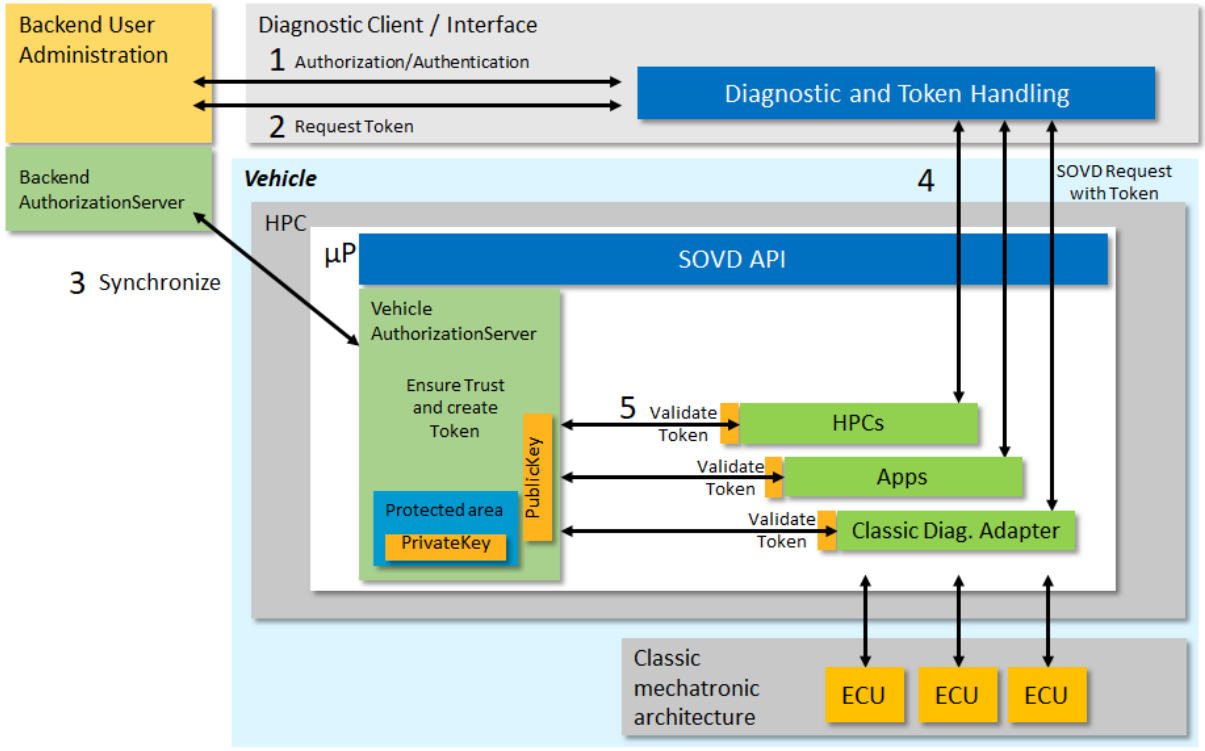

Figure 9: Online authentication and authorization

6.15.1.2 Offline Authentication and Authorization

An example for a typical sequence can be as shown in Figure 10:

1. The SOVD client needs to be authorized prior to be able to request a token at the vehicle by providing credentials (e.g., a certificate) to the Vehicle AuthorizationServer via the SOVD API. The Vehicle AuthorizationServer responds with an authorization code.
2. The SOVD client requests a token from the Vehicle AuthorizationServer and provides the authorization code as part of the token request. The SOVD client also provides its credentials in this step.
3. If allowed, the Vehicle AuthorizationServer issues a token to the SOVD client.
4. The SOVD client uses this token when issuing requests to the SOVD API.
5. The addressed SOVD server validates the token from the Vehicle AuthorizationServer (public key fitting the private key which was used to create the signature for the token) and the access can be granted.

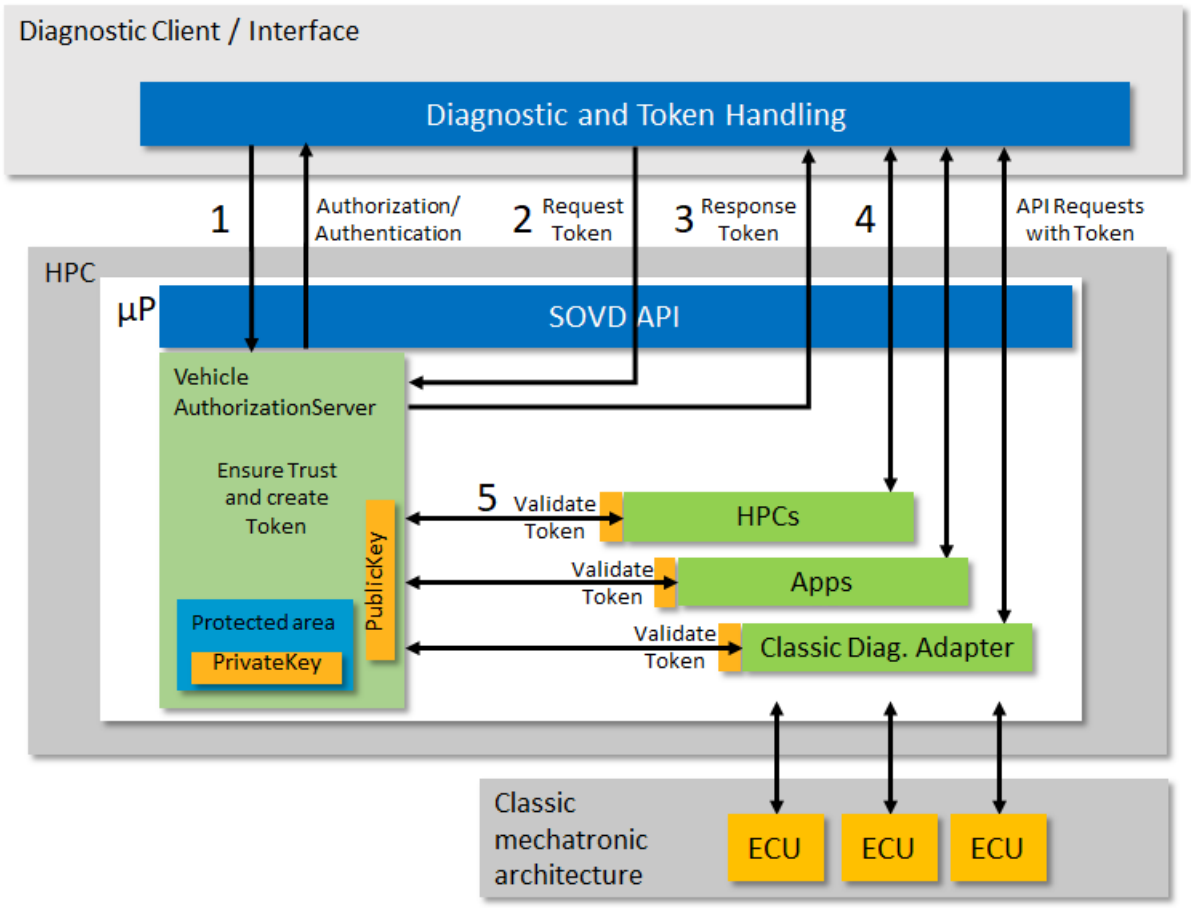

Figure 10: Offline authentication and authorization

6.15.1.3 Implementation Hints

Since the SOVD API shall be independent and non-restrictive regarding applicable authentication mechanisms and protocols for implementing SOVD client authorization as well as token-based authentication of requests, in the following, a basic set of resources is described with respective examples. An implementation of the SOVD API may provide additional resources (which are not standardized by the SOVD API) based on the selected authentication protocol and token format/implementation.

To model complex chained API calls (application calls API 1, and this API 1 calls another API 2 to fulfil a certain function), for both online and offline, the Vehicle AuthorizationServer could offer a mechanism called token exchange, to allow API 1 to act on behalf of the original calling application in regard to API 2. Effectively, API 1 can call API 2 with the permissions granted to the application. The technical specifics of this flow are defined in [12].

6.15.2 Secure Connection using TLS

The communication between the SOVD client and the AuthorizationServer and to any SOVD server as well as the communication inside the vehicle to the Vehicle AuthorizationServer or between each SOVD server requires a secure end-to-end channel, so that the communication cannot be intercepted or sniffed by any third party. For this, current consensus standards should be employed (e.g., at least TLS 1.2).

6.15.3 Verifying SOVD Client Credentials and Requesting a Token at the Backend

In the online authentication and authorization shown in Figure 9, an SOVD client which requires access to any SOVD server needs to be known and administrated in the backend. After successful authentication at the backend, the SOVD client can request a token at the Backend AuthorizationServer. If allowed, the Backend AuthorizationServer issues a token to the SOVD client. The Backend AuthorizationServer and the Vehicle AuthorizationServer build on established trust, i.e., the backend authenticates against the vehicle, and both AuthorizationServer instances are synchronized. Therefore, the Backend AuthorizationServer synchronizes the issued token created for the SOVD client with the Vehicle AuthorizationServer to enable token validation within the vehicle.

6.15.4 Verifying SOVD Client Credentials at the Vehicle

Method:

```
POST /authorize
```

Method Description:

This method initiates an authorization flow to authenticate and authorize an SOVD client to act upon a human user. Depending on the actual flow type, the result of this flow is an authorization code, that the client can use to obtain a token using a subsequent call to the /token API method, or an access token. This method is only used in the context of human user interaction.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Potential query parameters are defined by the selected authorization protocol.

Request Headers:

Potential request headers are defined by the selected authorization protocol.

Request Body:

The request body depends on the selected authorization protocol and its implementation by the SOVD API.

Response Status Codes:

Table 206 Response Status Codes - Verifying SOVD client credentials at the vehicle

Status Code	Response Body	Description
200	Response Body	The request was successful.
400	GenericError	If the SOVD client credentials are not provided as part of the request. The error code complies to the errors defined in [13].

Response Body:

The response body depends on the selected authorization protocol and its implementation by the SOVD API.

Example:

As the request and response behavior is highly dependent on the utilized, protocol, no example is given here.

6.15.5 Requesting a Token

Method:

```
POST /token
```

Method Description:

The SOVD client can request a token via this method. The AuthorizationServer shall accept requests to create tokens from authenticated SOVD clients only. Therefore, an SOVD client shall authenticate against the AuthorizationServer or have some other credentials to proof its identity prior a token is created and returned via the SOVD API. An SOVD client, which cannot be authorized shall not be able to request a token through the SOVD API. Traditional methods like client-id / client-secret combinations can be used, as well as advanced methods for instance, based on client certificates.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Potential query parameters are defined by the selected authorization protocol.

Request Headers:

Potential request headers are defined by the selected authorization protocol.

Request Body:

The request body depends on the selected authorization protocol and token implementation.

Response Status Codes:

Table 207 Response Status Codes - Requesting a token

Status Code	Response Body	Description
200	Response Body	The request was successful.
400	GenericError	If the challenge was not responded to correctly or something else is missing or is invalid (request, client-id, etc.) For example, if using OAuth 2.0 the following list of errors might be replied in the response body: [13] chapter 5.2

Response Body:

The response body depends on the selected authorization protocol and token implementation.

Example:

As for example for a token request, in the following, OAuth 2.0 [13] is assumed as underlying authorization protocol and “client credentials” are used as authorization grant. Example is taken from [14], chapter 4. Cache control for HTTP/2 should be deactivated in order not to cache secrets.

Request:

```
POST {base_uri}/token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLm
NvbS9pc19yb290Ijp0cnVlfQ.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk",
  "token_type":"Bearer",
  "expires_in":3600
}
```

6.15.6 Request Header for Access-Restricted Resources

When trying to access any access-restricted resource at the SOVD API, the SOVD client has to authorize against the server by providing a corresponding token in the request header.

Whenever an SOVD client requests data, a functionality or method from an SOVD server by sending an HTTP request to a URI of the SOVD API, a valid token shall be provided within the HTTP Authorization header of the request, e.g., by providing a Bearer [14] token [11]. If a token is valid or not is validated by the SOVD server with the help of the Vehicle AuthorizationServer as described in 6.15.7. This includes for example to check if the SOVD client sending the request is allowed to/has the required permissions to execute an HTTP method on the protected resource specified through the request.

The following response status codes are foreseen to provide information about authorization errors.

Table 208 Response Status Codes - Request header for access-restricted resources

Status Code	Response Body	Description
`401`	`--`	If no token is provided in the `Authorization` header of the request.
`403`	GenericError	If the token validation done by the SOVD server responsible for the target resource has failed. For example, the SOVD client identified by the token does not have the required access rights or role associated and therefore the token is not sufficient to interact with the protected resource using the selected HTTP method. For example, the SOVD client has not the access rights to delete the resource. If for security reasons the presence of a protected resource shall not be unveiled, the response code „404 Not Found“ can be sent. This principle is also described in the HTTP standard.

6.15.7 Validating a Token

If the SOVD API receives a request by an SOVD client, the accompanying token will be validated by the SOVD API. The validation itself is conducted by the underlying SOVD server with the help of the Vehicle AuthorizationServer. The SOVD server has to request the public key from the Vehicle AuthorizationServer to validate the signature of the token as well as any other limiting content in the token. Based on that, the SOVD server will determine whether the SOVD client is authorized to execute the request. Depending on the result of the validation, the requested resource can be accessed or the SOVD server rejects the request with a respective HTTP response status code within the response to indicate that the validation was not successful as described in 6.15.6.

7 Classic Diagnostic Adapter

7.1 Introduction

The introduction of SOVD will not eliminate classic diagnostic protocols such as UDS in the vehicles. Furthermore, it is assumed that a considerable number of devices will still be based on UDS diagnostics. Therefore, the SOVD API can also provide access to components which are based on classic diagnostic protocols.

Typically, a component in the SOVD Server will perform the mapping of the SOVD HTTP requests to the classic diagnostic protocol. It is not foreseen that the classic components themselves will provide an SOVD interface.

Currently the standard only describes the mapping of SOVD to UDS, however, other mappings can be implemented.

It is not mandatory for an SOVD server to expose classic diagnostic services as resources. If it doesn't, it is assumed that the server handles these classic diagnostic services internally (e.g., ensuring correct session/security levels) and an SOVD client does not need access to them. If the SOVD API provides individual access to the classic diagnostic services, it shall be done as specified in this chapter.

7.2 Access to UDS Based Entities

There is an overlap between the API methods supported by SOVD and the diagnostic services provided via UDS. Therefore, the approach of SOVD is to include the UDS based content into the defined API methods. There are no special API methods defined for accessing UDS content.

Table 209 Overview on how to map UDS based contents to SOVD

Service-Id	REST resource	See
0x10	<entity>/modes/**session**	7.3.1
0x11	<entity>/operations/**ecureset**	7.6.2
0x27, 0x29	<entity>/modes/**security**	7.3.2
0x28	<entity>/modes/**commctrl**	7.3.4
0x3E	Will be handled by the Classic Diagnostic adapter	
0x85	<entity>/modes/**dtcsetting**	7.3.5
0x22, 0x2E	<entity>/data/ <entity>/configurations/	7.4
0x14, 0x19	<entity>/faults/	7.5
0x2F	<entity>/operations/	7.6.1
0x31	<entity>/operations/	7.6.1

Service-Id	REST resource	See
0x23, 0x24, 0x2A, 0x2C, 0x34, 0x35, 0x36, 0x37, 0x38, 0x84, 0x86	Will not be represented on SOVD API level	

7.3 Specific Mapping of UDS Services to SOVD Modes

7.3.1 Mapping of SessionControl ($10) Subfunctions to SOVD Modes

There shall be a mode **session** representing the available UDS diagnostic session in SOVD. The **`PUT`** method (see 6.10.4) shall support an ECU-specific enumeration representing the subfunctions of service $10 supported by the UDS ECU. This method may also control the sending of TesterPresent commands by switching to a non-default session.

7.3.2 Mapping of SecurityAccess ($27)

This subchapter provides advice how UDS security using service $27 could be implemented in an SOVD server.

UDS Service $27 requires a security access sequence. This functionality can be

a) performed as a sequence implicitly by the SOVD server when an SOVD mode **security** is established,
b) performed as a sequence implicitly by the SOVD server when a UDS ECU requiring $27 security is accessed or
c) provided by exposing the individual $27 subservices as operations at the SOVD API (the sequence has to be implemented in the client in this case).

If option c) is used, a mode **security** representing the available UDS security access modes in SOVD can be implemented.
The **`GET`** method (see 6.10.3) supports an ECU-specific enumeration representing the RequestSeed subfunctions of service $27 supported by the UDS ECU.
The **`PUT`** method (see 6.10.4) supports an ECU-specific enumeration representing the SendKey subfunctions of service $27 supported by the UDS ECU.

7.3.3 Mapping of $29 Authentication to SOVD Modes (Informative)

This subchapter provides advice how UDS security using service $29 could be implemented in an SOVD server.

UDS Service $29 requires an authentication sequence. This functionality can be

a) performed as a sequence implicitly by the SOVD server when an SOVD mode **authentication** is established,
b) performed as a sequence implicitly by the SOVD server when a UDS ECU requiring authentication is accessed or
c) provided by exposing the individual $29 subservices as operations at the SOVD API (the sequence has to be implemented in the client in this case).

Options a) and b) provide the possibility to handle the authentication within the SOVD server, while option c) exposes the subservices as defined in ISO14229-1 as used in the respective vehicle.

7.3.4 Mapping of Communication Control ($28) Subfunctions to SOVD Modes

There shall be a mode **comm-ctrl** representing the communication control service.
The **`PUT`** method (see 6.10.4) shall support an ECU-specific enumeration representing subfunctions of service $28 supported by the UDS ECU.

7.3.5 Mapping of Control DTC Settings ($85) Subfunctions to SOVD Modes

There shall be a mode **dtcsetting** representing the control DTC setting service.
The **`PUT`** method (see 6.10.4) shall support an enumeration (“on/off”) representing UDS subfunctions of service $85 supported by the UDS ECU.

7.4 Mapping of UDS Services to Data Resources

UDS data identifiers available via ReadDataByIdentifier ($22) shall be available as data resources in SOVD. An SOVD server could determine the category of data based on its input data (e.g., ODX semantics). It is recommended to provide an individual data resource for each UDS data identifier.
In case a data identifier is also writable (using WriteDataByIdentifier $2E) it shall be available via the same data resource using the **`PUT`** method.
Alternatively, data identifiers for writing can also be provided as configurations resources.

7.5 Mapping of UDS Services to Fault Resources

The DTC information available via UDS Services $19 and $14 shall be available as individual faults in the faults resource collection representing the UDS entity.
See Table 46 for details on which fields in the SOVD API resemble fault information provided for classic ECUs.

7.6 Mapping of UDS Services to Operation Resources

7.6.1 Mapping of Routine and IOControl Services to Operation Resources

IOControl Identifiers (available via Service $2F and $22) and Routine Identifiers (available via Service $31) shall be available as operations resources.
For the relation of IOControl and RoutineControl UDS subfunctions to API methods see Table 102.

Note: Several UDS services (with different UDS subfunctions) referring to the same Routine/IOControl Identifier shall be mapped to the same operations resource.

7.6.2 Mapping of EcuReset Service ($11) to Operation Resources

In case the UDS entity supports ECUReset, there shall be a resource **ecureset** in the resource collection for operations. The resource shall support a start parameter “ResetType” being an ECU-specific enumeration of the different UDS subfunctions supported by the ECU.

8 Terms and Definitions

For the purposes of this ASAM standard the following terms and definitions apply (in alphabetical order).

Authentication — It is the mechanism employed to verify an identity towards a software system. Within SOVD authentication is used to verify the identity of the SOVD client (user).
Remark: Examples for authentication method are ISO 14229-1 Service $29 [1], TLS 1.3. [3], IEEE 1609.2[7] or X.509 [8]-Attribute Certificates

Authorization — Is the mechanism to verify whether an authenticated SOVD client (user) has sufficient privileges to execute a specific SOVD method. It is also used to verify, whether an authenticated SOVD client (user) has sufficient privileges to access specific data or capabilities of the vehicle that can be obtained or invoked by executing a specific SOVD method (e.g., it is possible a user has access to SOVD method A, but is only allowed to retrieve a subset of the data that can potentially be obtained by calling SOVD method A)

Bulk Data — Specifies any larger set of structured or unstructured digital data. The volume of the data is high and / or in proprietary or binary format, so that it is not possible or desirable to standardize its format/structure specifically as part of a request/response structure of a SOVD method.

Capability — Describes a possibility of interaction with an entity via SOVD. A capability can be a specific method, a sensor value, a routine, a coding parameter, an access to log files etc.

Capability Description — Is a human- and machine-readable record of all capabilities of a specific vehicle (instance-specific CD) or a set of similar vehicles, e.g., a vehicle platform (general CD).
Remark: As such, it resembles an ODX-file for a classic ECU.

Note: A capability description does not have to be statically written to a vehicle or entity. Rather, they can be composed at runtime from various data sources. Information that may later contribute to an in-vehicle capability description generation, can be transferred into the vehicle (or into entities) via existing methods like bulk data writing, vehicle / entity configuration, vehicle / entity reprogramming. It may be specific to OEMs processes, vehicle architecture and update process considerations, which way to handle the capability description may be appropriate.

Credentials	Are used by software modules to control access to information or other resources. The classic combination of a user's account number or name and a secret password is a widely used example of IT credentials. An increasing number of information systems use other forms of documentation of credentials, such as biometrics (fingerprints), public key certificates, and so on.
Diagnostic Trouble Code	Fault or event that was stored by an ECU as part of its self-diagnostic capabilities.
ECU	Electronic Control Unit – an embedded computing device within the vehicle that is mainly used for control functions and is based on real-time oriented operating systems like OSEK [4], AUTOSAR classic [5].
HPC	High Performance Computer – a powerful computing device within the vehicle that is capable of hosting multiple virtualized guest systems based on operating systems.
In-Vehicle Diagnostics	Diagnostic applications that are self-sustained and executed within the vehicle (potentially running on an HPC), which means they do not require a connection to the Internet to be executed.
Offline Capability Description	Is an offline-version of a capability description that exists outside of the context of a particular vehicle (e.g., as a file, database data).
Online Capability Description	Is a version of a capability description that is obtained directly from a vehicle instance or an HPC through SOVD capability discovery mechanisms.
Operation	An operation is an internal function, which shall be controlled and parameterized by the SOVD API. Operations are used additionally as synonym for actuators (I/O control) and routines.
Proximity Diagnostics	Diagnostic applications that are executed on an external device (i.e., device that is not part of the vehicle) that is connected to an SOVD server of a vehicle in a wired or wireless way using common LAN or NFC technology.
Remote Diagnostics	Diagnostic applications that are executed in a remote external environment (e.g., backend, cloud) that is connected to the vehicle using common WAN technology.
Resource	The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g., a person), and so on. REST uses a resource identifier to identify the resource involved in an interaction between components.

9 Symbols and Abbreviated Terms

API	Application Programming Interface
APP	APPlication
C	Conditional
CD	Capability Description
CDA	Classic Diagnostic Adapter
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DID	Datal IDentifier
DLT	Diagnostic Log and Trace
DNS	Domain Name System
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
E/E	Electrical / Electronic architecture
FOTA	Firmware Over The Air
HPC	High Performance Computer
HTTP	HyperText Transfer Protocol
ID	IDentifier
IETF	Internet Engineering Task Force
IT	Information Technology
JSON	Java Script Object Notation
LAN	Local Area Network
M	Mandatory
MD5	Message-Digest Algorithm 5
MIME	Multipurpose Internet Mail Extensions
NFC	Near Field Communication
O	Optional
ODX	Open Diagnostic data eXchange
OEM	Original Equipment Manufacturer
OS	Operating System
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
OTA	Over The Air

REST	REpresentational State Transfer
RFC	Request For Comments
SD	Service Discovery
SOVD	Service-Oriented Vehicle Diagnostics
TLS	Transport Layer Security
UDS	Unified Diagnostic Services
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
VDX	Vehicle Distributed eXecutive
VIN	Vehicle Identification Number
WAN	Wide Area Network
XML	eXtensible Markup Language
YAML	Yet Another Markup Language

10 Bibliography

[1] ISO 14229-1:2020 Road vehicles — Unified diagnostic services (UDS) — Part 1: Application layer

[2] IETF RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content; 2014

[3] IETF RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3; 2018

[4] ISO 17356-3:2005 Road vehicles — Open interface for embedded automotive applications — Part 3: OSEK/VDX Operating System (OS)

[5] AUTOSAR; Classic Platform Release Overview

[6] AUTOSAR; Specification of Diagnostic Log and Trace

[7] IEEE 1609.2 - IEEE Standard for Wireless Access in Vehicular Environments--Security Services for Applications and Management Messages

[8] ISO /IEC 9594-08:2017 Information technology — Open Systems Interconnection — The Directory — Part 8: Public-key and attribute certificate frameworks

[9] ISO 22901-1:2008 - Open diagnostic data exchange (ODX) - Part 1: Data model specification

[10] Open API Initiative; OpenAPI Specification v3.1.0; 2021

[11] IETF RFC 7519; JSON Web Token (JWT); 2015

[12] IETF RFC 8693; OAuth 2.0 Token Exchange; 2020

[13] IETF RFC 6749; The OAuth 2.0 Authorization Framework; 2012

[14] IETF RFC 6750; The OAuth 2.0 Authorization Framework: Bearer Token Usage; 2012

[15] IETF RFC 1847; Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted; 1995

[16] R. T. Fielding; "REST: Architectural Styles and the Design of Network-based Software Architectures," PhD dissertation, University of California, Irvine, 2000

[17] Ecma international; Standard ECMA 262 ECMA Script 2021 Language Specification; 2021

[18] ISO 22900-2; Road vehicles — Modular vehicle communication interface (MVCI) — Part 2: Diagnostic protocol data unit application programmer interface (D-PDU API); 2009

[19] IETF RFC 3339; Date and Time on the Internet: Timestamps; 2002

[20] IETF RFC 6762; Multicast DNS; 2013

[21] ISO 8601-1; Date and Time Representations for information exchange – Part 1: Basic rules: 2019

[22] IETF RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2); 2015

[23] IETF RFC 2616; Hypertext Transfer Protocol Version 2 (HTTP/1.1); 1999

[24] IETF RFC 6763; DNS-Based Service Discovery; 2013

[25] JSON; JSON schema specification; 2020

[26] ASAM e.V.; ODX Authoring Guidelines; 2011

[27] IETF RFC 1341; MIME (Multipurpose Internet Mail Extensions); 1992

[28] IETF RFC 8259; The JavaScript Object Notation (JSON) Data Interchange Format; 2017

[29] ISO 22900-3; Road vehicles — Modular vehicle communication interface (MVCI) — Part 2: Diagnostic server application programming interface (D-Server API); 2012

[30] IETF RFC 5322; Internet Message Format; 2008

[31] IETF RFC 3986; Uniform Resource Identifier (URI): Generic Syntax; 2005

[32] IETF RFC 6570; URI Template; 2012

[33] IETF RFC 6901; JavaScript Object Notation (JSON) Pointer; 2013

[34] IETF RFC 4122; A Universally Unique IDentifier (UUID) URN Namespace; 2005

[35] ASAM e.V.; ASAM Data Types; 2005

[36] IETF RFC 5424; The Syslog Protocol; 2009

Appendix: A. SOVD API Common Example

Informative

A.1. Introduction

In this appendix, an example is given for the SOVD API. It is not intended to show a complete example here, but with enough details to explain the intended usage of the SOVD API. This example is used as the base for the method examples given in the standard.

For this example, object diagrams are used to describe the entity hierarchy and available resources for each entity. For more complex sequences of methods defined in the SOVD API, sequence diagrams are provided.

The diagrams in this example use the UML notation. The object diagrams show the relation between the entities themselves, and between entities and resources. These diagrams are not to be understood as a model of the underlying data in the vehicle.

Furthermore, this appendix provides an example of an OEM specific realization of a ***Function*** entity and how this can be documented.

A.2. Entity Hierarchy

Figure 11 shows the entity hierarchy of the example for an SOVD server named “Root”. The SOVD server includes one top-level ***Area*** “Driving”, which contains the two subareas “Perception” and “Controlling”. Each of these ***Area***s link to top-level ***Component***s under “Root”.

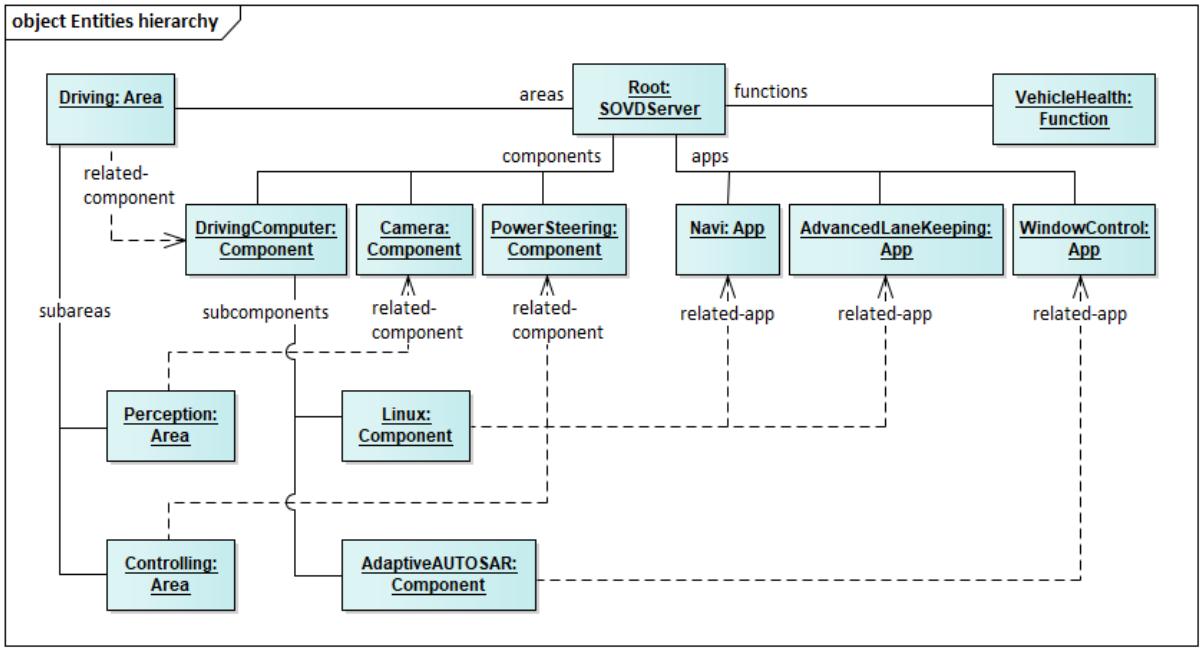

Figure 11 Example hierarchy for “Root”

The ***Component*** “DrivingComputer” is an HPC running a hypervisor on which two different operating systems are executed. Both OSs are integrated as subcomponents of “DrivingComputer” into the SOVD API hierarchy and link to ***App***s. The other ***Component***s “Camera”, and “PowerSteering” are defined as embedded ECUs in this example.

A.3. Entities and Their Resources

The following object diagrams illustrate the resource collections and resources used in the example HTTP snippets in the SOVD API standard.

A.3.1. App AdvancedLaneKeeping

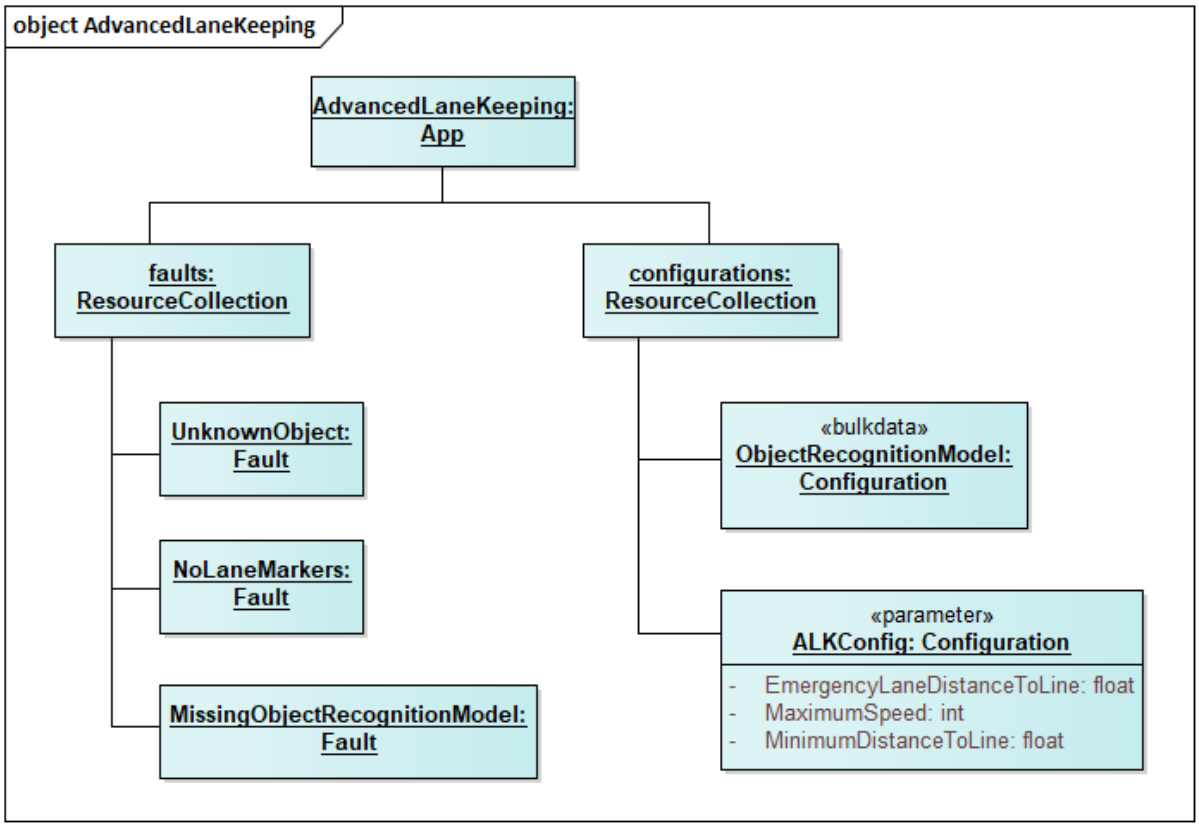

Figure 12 Resources of the App “AdvancedLaneKeeping”

A.3.2. App WindowControl

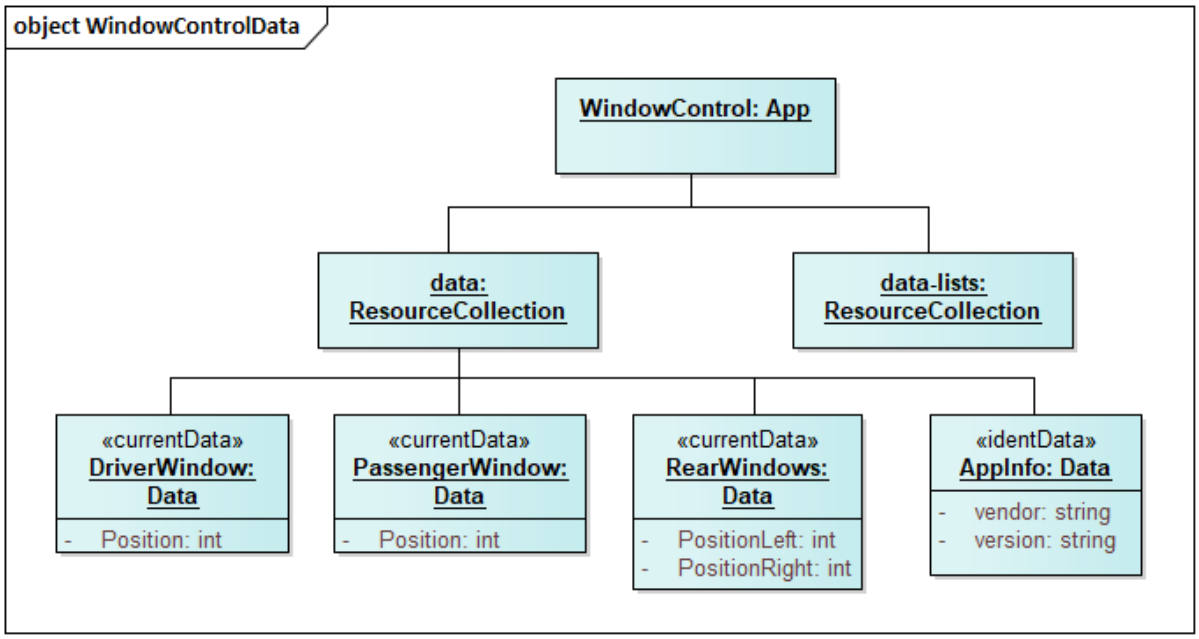

Figure 13 Resources of the App "WindowControl"

A.3.3. App Navi

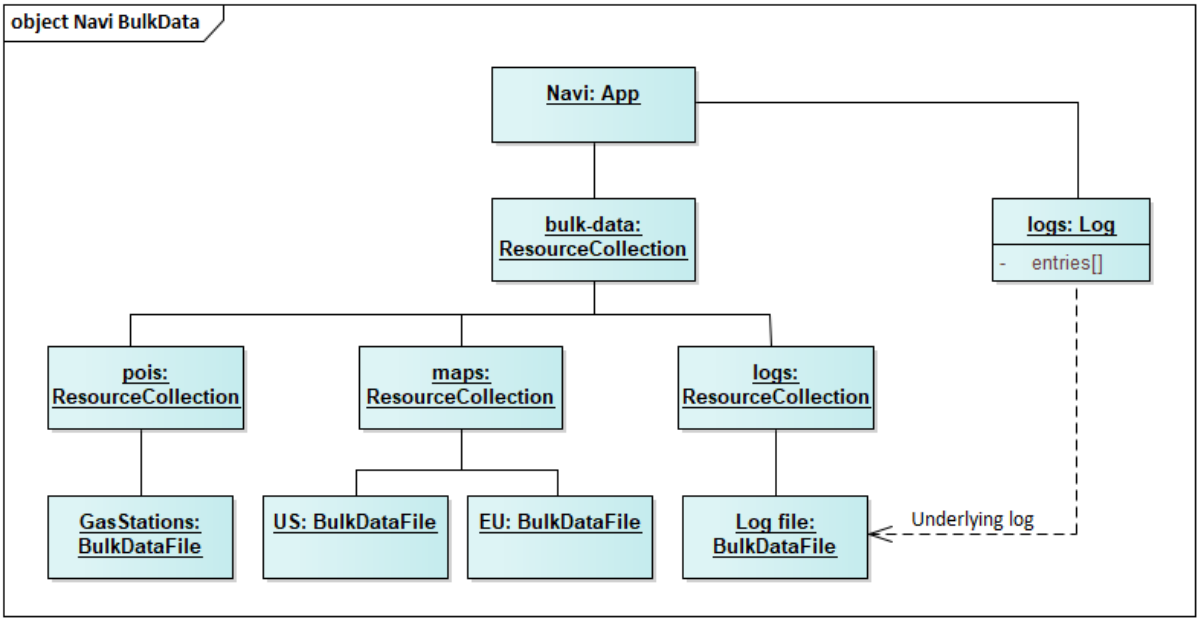

Figure 14 Resources of the App "Navi"

A.3.4. Component Camera

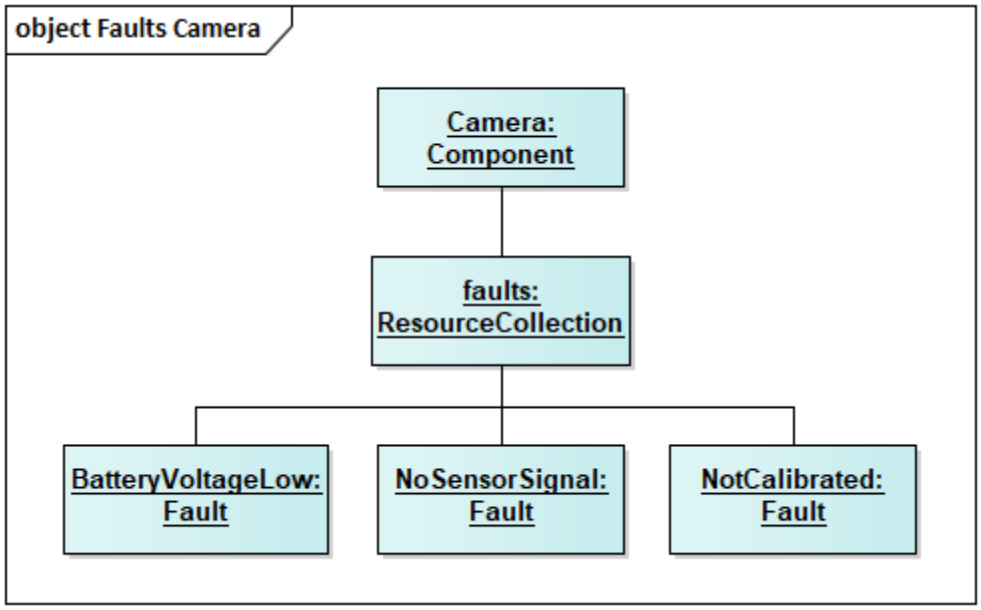

Figure 15 Resources for the Component “Camera”

A.3.5. Component PowerSteering

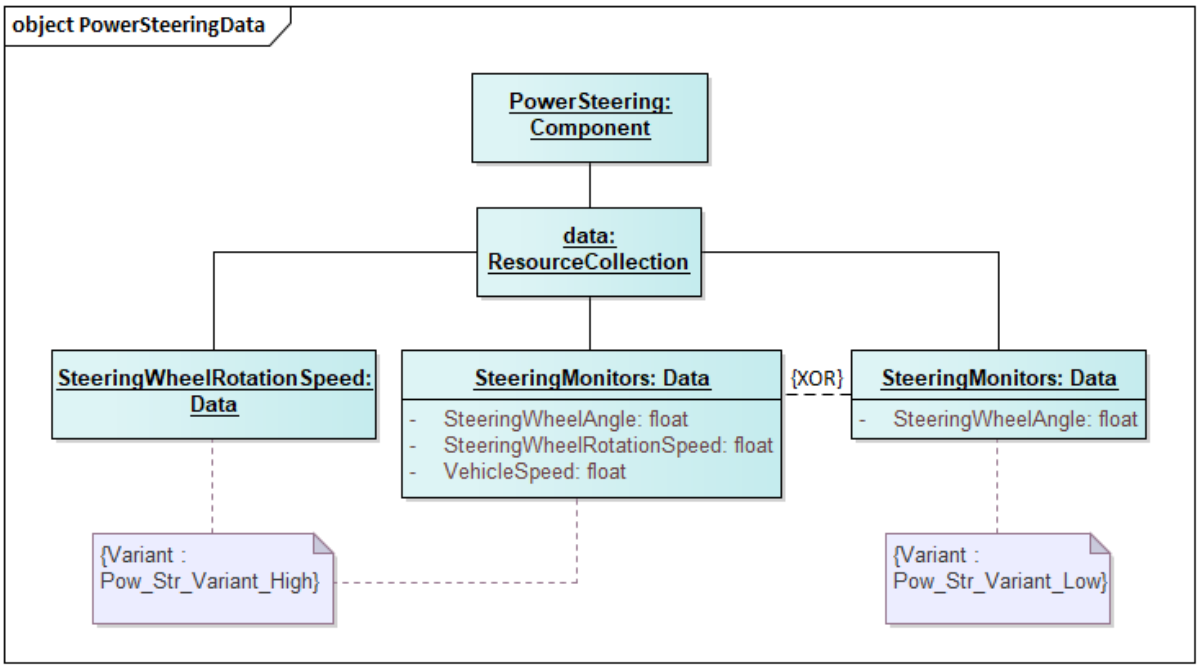

Figure 16 Resources and variants for the Component “PowerSteering”

A.4. Complex Sequence Examples

The following subchapters show the usage of the SOVD API in situations where a sequence of API methods has to be called. These examples are in line with the HTTP snippets at the respective API methods in chapter SOVD REST API.

A.4.1. Creation and Usage of a Temporary Data-List Resource

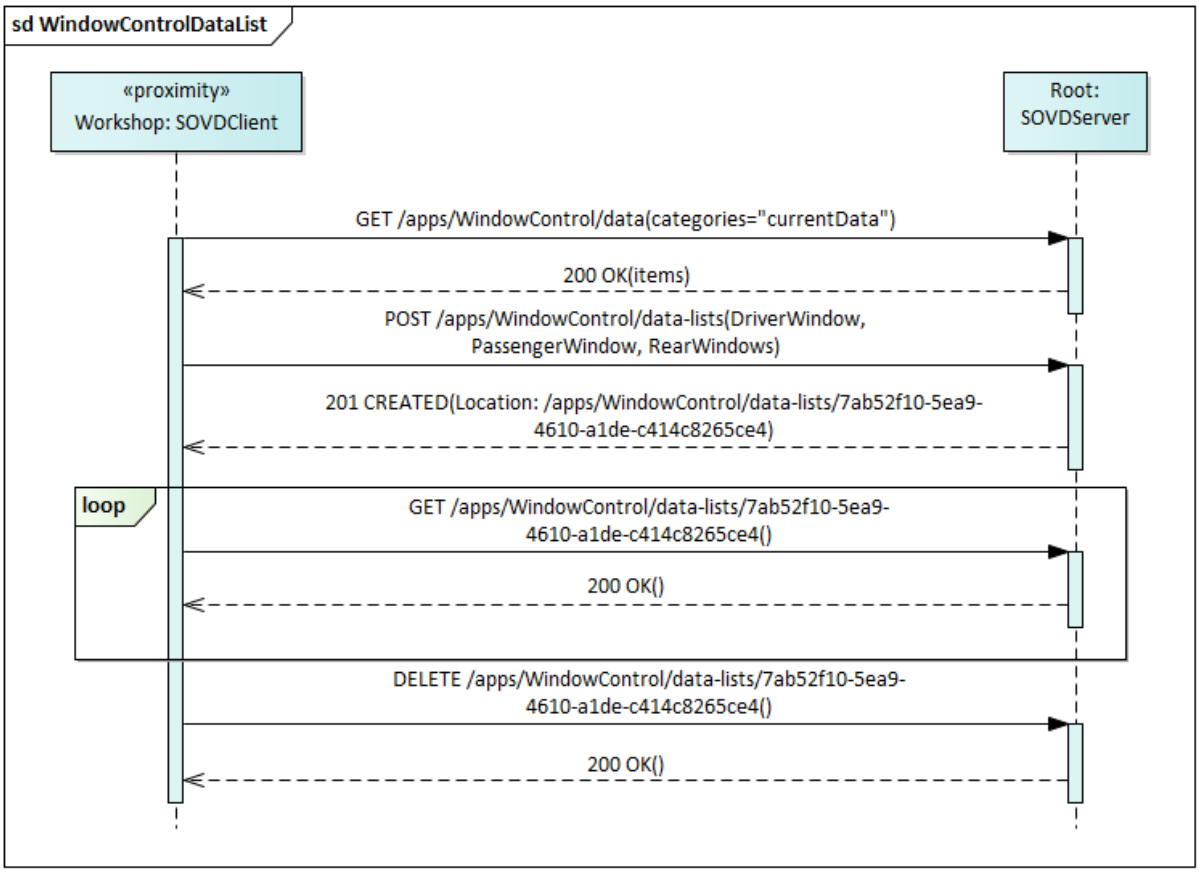

Figure 17 UML sequence diagram for a data-list resource

A.4.2. SteeringAngleControl Operation incl. Modes and Locks

The following subchapter gives an overview how the execution of an operation via the SOVD API may look like for an “I/O-control” operation (SteeringAngleControl) of a classic ECU (PowerSteering) as an example. Therefore, also required prerequisites to execute this operation are described in the following. This comprises locking the underlying ***Component*** to prevent any interference with the execution of other operations or parallel access of other SOVD clients as well as manually setting the ECU to a specific mode which is required to execute the operation. For each of the steps, the methods introduced within this document are used as a basis. Respective examples for the underlying HTTP requests and responses are given in the API method descriptions.

Figure 18 depicts an excerpt of the topology, i.e., the SOVD API resources, relevant for the execution of the example operation “SteeringAngleControl” at the respective ECU “PowerSteering”. The operation allows to control the steering angle in two different ways. With the parameter “control-type” set to “absolute”, the PowerSteering controls the steering angle

to the position specified in parameter “angle”. With the “control-type” set to “relative”, the PowerSteering will adjust the current angle by the value provided by the parameter “angle”. We assume that the “SteeringAngleControl” operation requires a lock and specifies that the “PowerSteering” has a “session” mode which has to be set to “EXTENDED” in order to execute the operation. The SOVD server support “Explicit” mode control as well as “Hint-based”. It is not capable of implicitly identifying the required mode setting based on the executed operation.

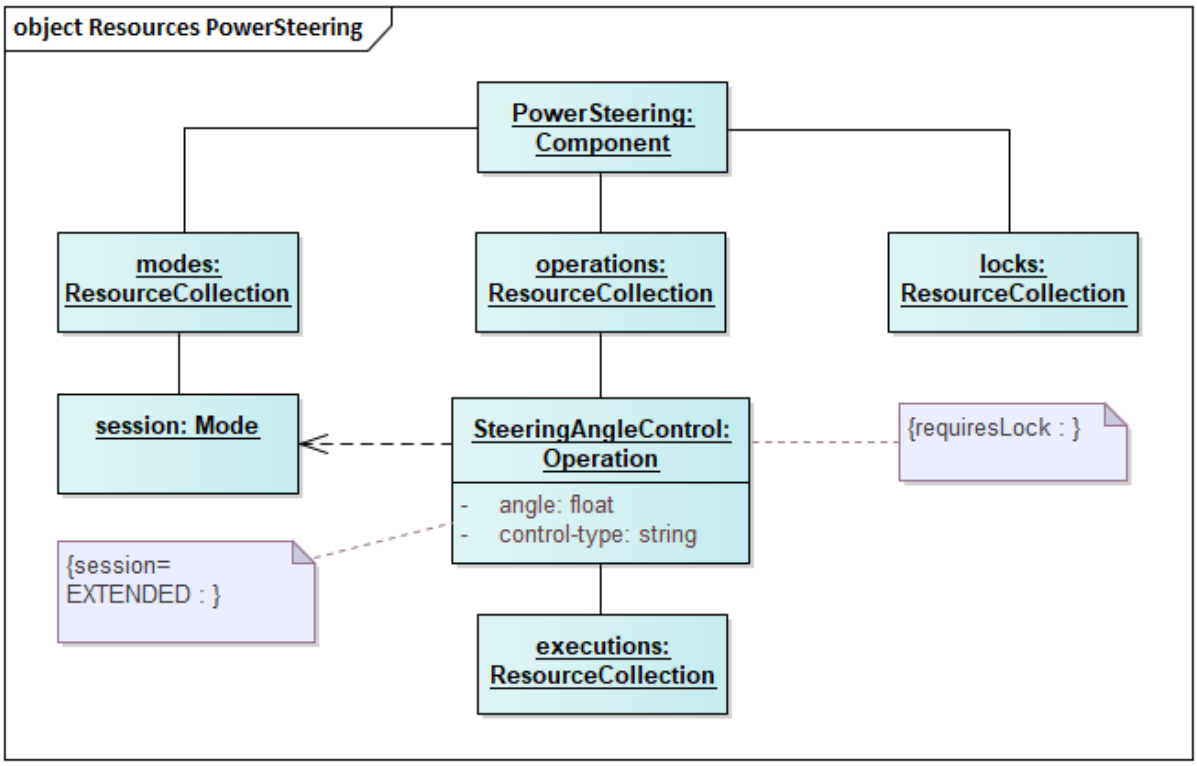

Figure 18: Topology for the “SteeringAngleControl” operation of “PowerSteering”

The overall sequence of steps for controlling an operation of an ECU are shown in Figure 19. For the sake of understandability, no exceptional or erroneous paths are presented within the sequence diagram nor discussed as part of the example in the following. Furthermore, the proximity challenge is not explicitly reflected as part of the sequence diagram.

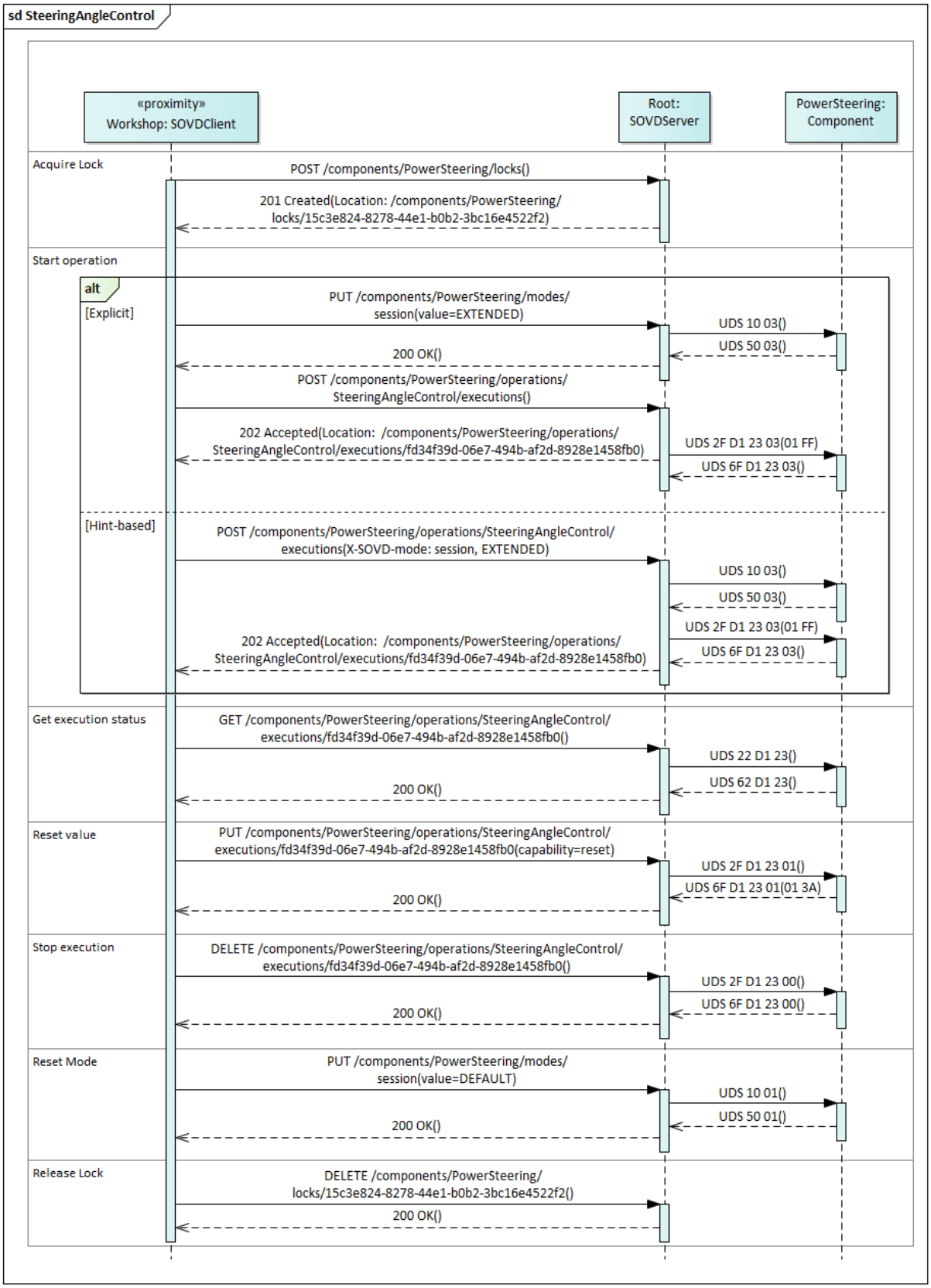

Figure 19: UML sequence diagram for the "SteeringAngleControl" operation

The following list summarizes the steps depicted in the sequence diagram above:

- **Acquire Lock:** The SOVD client acquires a lock of the ECU which allows the operation to be executed.
- **Start Operation [Explicit]:** The SOVD client explicitly sets the required mode “session” to “EXTENDED”, which is a prerequisite for executing the selected operation. The SOVD client requests the execution of the selected operation with a **`POST`** request containing all required input parameters and further information. The SOVD server then triggers the execution of the respective operation in the “PowerSteering” ***Component*** and creates a new execution resource. In this alternative, the SOVD server has identified that the operation can be executed and responds to the SOVD client before the ECU sends a positive response. The SOVD client has to request the status to learn that the ECU may have sent a negative response.
- **Start Operation [Hint-based]:** As an alternative to the two previous steps, the SOVD client requests the execution of the selected operation with a **`POST`** request and provides the “hint” to the SOVD server to set the mode “session” to “EXTENDED”. This request also contains all required input parameters. The SOVD server sets the requested mode and triggers the execution of the respective operation in the “PowerSteering” ***Component*** and creates a new execution resource. Here, the SOVD server waits for a positive response from the ECU before sending the answer to the SOVD client. This behavior is not specified and left to the SOVD server’s implementation.
- **Get execution status:** This temporary execution resource is then used by the SOVD client to poll the status of the operation execution via a **`GET`** request. As soon as the SOVD server receives a result from the ECU “PowerSteering” for the operation execution, the result data is returned to the SOVD client.
- **Reset value:** Before the operation is stopped, the SOVD client requests to reset the value of the operation via a **`PUT`** request and the capability set to “reset”.
- **Stop execution:** Afterwards, the SOVD client stops the operation via a **`DELETE`** API method.
- **Reset Mode:** The SOVD client deactivates the previously activated mode of the ECU, e.g., by setting the mode “session” to “DEFAULT”.
- **Release Lock:** Finally, the SOVD client releases the acquired lock on the ECU to enable others SOVD clients to interact with the previously locked resources again.

A.5. Function VehicleHealth

A.5.1. Introduction

In this part, an example is provided for a ***Function*** entity. It shows how an OEM can define ***Function***s and describe the API methods available.

Since the SOVD API does not permit the retrieval of aggregated faults for an ***Area*** and thus all related ***Component***s, this example realizes this functionality in an SOVD server using an entity type ***Function***. Figure 20 shows the ***Function*** `VehicleHealth` beneath an SOVD server `VehicleDiag`.

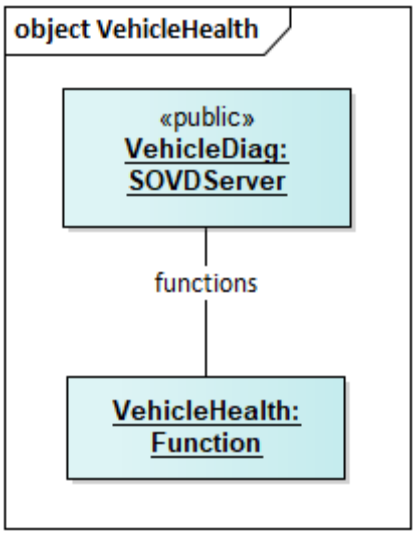

Figure 20 Function VehicleHealth

The ***Function*** described in this annex collects the faults – either just the fault count or all the details – as well as the hardware and software versions of all ***Component***s in the vehicle.

For the sake of simplicity, the following assumptions are made:

- Each ***Component*** is able to provide the collected information upon request in a timely manner. Thus, timeouts due to long running request or asynchronous responses are not considered.
- In a multiple SOVD server environment the method collects the vehicle health from other SOVD servers.

A.5.2. Retrieval of VehicleHealth

Method:

```
GET /functions/VehicleHealth
```

Method description:

The ***Function*** gathers the health of a vehicle based on the SOVD API as defined in the introduction.

Path Parameters:

This method does not support path parameters.

Query Parameters:

Table 210 Query Parameters - Retrieval of VehicleHealth

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

`fault-details`	`boolean`	O	Specifies whether the response should include details on the fault or just the number of faults. The default is `false`.

Response Status Codes

Table 211 Response Status Codes – Retrieval of VehicleHealth

Status Code	Response Body	Description
`200`	Table 212	The request was successful.
`502`	GenericError	The SOVD server was unable to retrieve vehicle health from any ***Component***, but the error was not in the SOVD server itself.

Response Body:

Table 212 Response Body - Retrieval of VehicleHealth

Attribute	Type	Convention	Description
`items`	VehicleHealth `[]`	M	Array with the vehicle health of each ***Component***.
`schema`	`Schema`	C	The schema definition of the response. Condition: Only provided if the query parameter `include-schema` is `true`.

Table 213 `VehicleHealth` type

Attribute	Type	Convention	Description
`component`	`string:uri-reference`	M	URI to the ***Component*** providing the information.
`numberOfFaults`	`number`	C	Number of faults stored for the ***Component***. Only returned if `fault-details` is `false`.
`faults`	Fault `[]`	C	Faults as returned by **`GET /{entity-path}/faults`** without setting any of the query parameters. Only returned if `fault-details` is `true`.
`swVersions`	`AnyValue`	C	Value of the software version(s). The type `AnyValue` allows to store multiple version information, e. g. the version of bootloader, application, and

			calibration data depending on the ECU.
`hwVersions`	`AnyValue`	C	Value of the hardware version(s).
`errors`	DataError`[]`	C	Error describing why a particular information of the vehicle health of a ***Component*** could not be retrieved. Only set if information could not be retrieved from a ***Component***.

If an error occurs, the `errors` attribute of the VehicleHealth is set. For each erroneous attribute, the field will contain an entry. The attribute path of an error uses a JSON pointer to reference the erroneous attribute with a more detailed error description. In case the path points to ***Component***, the vehicle health for the ***Component*** could not be retrieved.

Appendix: B. Specific Variant Applicability

Informative

B.1. Introduction

The author can define which response or request schema is applicable for specific variants using the `x-sovd-applicability` extension. This is achieved by first using `oneof` to specify only one of a set of schemas would match. And in each schema, the `x-sovd-applicability` specifies the variant(s) for which the corresponding schema is applicable.

In the below example, **`/components/PowerSteering/data/SteeringMonitors`** resource is defined to retrieve some of the parameters of PowerSteering ECU. Based on the variant of PowerSteering mounted, two different response schemas are possible – VariantHighResponse and VariantLowResponse. The variant “Pow_Str_Variant_High” provides information of three data elements (VehicleSpeed, SteeringWheelRotationSpeed, SteeringWheelAngle) while the variant “Pow_Str_Variant_Low” provides information of only one data element (SteeringWheelAngle).

B.2. Example

```
"paths": {
  "/components/PowerSteering/data/SteeringMonitors": {
      "get": {
        "summary": "Gets the diagnostic information about power
                    steering",
        "responses": {
          "200": {
            "description": "description of successful response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/inline_response_200"
                }
              }
            }
          },
          "400": {
            "description": "bad input parameter"
          }
        }
      }
    },
"components": {
    "schemas": {
      "VariantHighResponse": {
        "type": "object",
        "properties": {
          "VechileSpeed": {
            "$ref": "#/components/schemas/VehicleSpeed"
          },
          "SteeringWheelRotationSpeed": {
```

```
        "$ref": "#/components/schemas/SteeringWheelRotationSpeed"
      },
      "SteeringWheelAngle": {
        "$ref": "#/components/schemas/SteeringWheelAngle"
      }
    },
    "description": "Response Schema Pow_Str_Variant_High",
    "x-sovd-applicability": {
      "description": "Applicability Extension to specify which
                      variant supports this schema",
      "variant_id_list": {
        "type": "array",
        "items": [
          "Pow_Str_Variant_High"
        ]
      }
    }
  },
  "VariantLowResponse": {
    "type": "object",
    "properties": {
      "SteeringWheelAngle": {
        "$ref": "#/components/schemas/SteeringWheelAngle"
      }
    },
    "description": "Response Schema Pow_Str_Variant_Low",
    "x-sovd-applicability": {
      "description": "Applicability Extension to specify which
                      variant supports this schema",
      "variant_id_list": {
        "type": "array",
        "items": [
          "Pow_Str_Variant_Low"
        ]
      }
    }
  },
  "VehicleSpeed": {
    "maximum": 200,
    "minimum": 0,
    "type": "integer",
    "example": 40
  },
  "SteeringWheelRotationSpeed": {
    "maximum": 30,
    "minimum": 0,
    "type": "integer",
    "example": 15
  },
  "SteeringWheelAngle": {
    "maximum": 270,
    "minimum": 0,
    "type": "integer",
    "example": 45
  },
```

Figure Directory

Table Directory

Email: support@asam.net
Web: www.asam.net