Document Title	Explanation of Service-Oriented Vehicle Diagnostics
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1064

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R25-11

Document Change History

Date	Release	Changed by	Description
2025-11-27	R25-11	AUTOSAR Release Management	• Changes in chapter for use case "SW Update"
2024-11-27	R24-11	AUTOSAR Release Management	• No content changes
2023-11-23	R23-11	AUTOSAR Release Management	• Update for R23-11 with alignment to supported SOVD use case
2022-11-24	R22-11	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction

In 2022 ASAM released the first version of the new Diagnostic SOVD Standard [1]. This document explains how SOVD is being realized by AUTOSAR and aims on giving use-case centric guidance. This document explicitly aims on going beyond the scope of the Functional Cluster ara::diag and wants to provide a more general picture to properly address the holistic character of SOVD.

The realization of SOVD seems only feasable on HPC machines due to the required technolgies such as HTTPS. Still the SOVD protocol is specified on vehicle level and therefore also raises the question how microcontrollers (typically diagnosed with UDS) can be included. Chapter 2 introduces a SOVD reference architecture, that will tackle these challenges. While incorporation of UDS within the SOVD API is a fundamental concept of the standard, SOVD also introduces new methods for diagnosing HPC machines. The native realization of these new methods by AUTOSAR Adaptive will be covered in chapter 3.

1.1 A Brief summary of ASAM SOVD

Since SOVD is a rather new Standard this section aims on giving a brief summary on the motivation for introducing a new standard as well as a short technical summary on the standard. This section will be on a general protocol level and therefore not focus on AUTOSAR specific realization.

1.1.1 Motivation

As of today (year 2023) the prevailing Diagnostic protocol of the automotive industry is UDS. UDS was initially standardized in 2006 as ISO 14229 and mainly focuses on diagnosing microcontrollers with hard ROM and RAM requirements. Since the protocol focuses on effiency, the interpretation of the data is handled on client side using ODX files. This raises challenges for the client technology stack and also means a hard dependency between client ODX file and the present Diagnostic implementation.

SOVD tackles these two challenges. The protocol is self-explaining, meaning it does not rely and has no dependency to an external ODX data description. Further SOVD is a "State-of-the-Art diagnostic API using current information technologies" [1]. This allows the client to rely on existing technology for the underlying protocols such as HTTPS.

Looking at todays vehicle architectures, HPC machines take over central functionality and allow to handle the increasing number of requirements on vehicle side. Also with the increasing software complexity new requirements on diagnostics methods are raised.

In summary ASAM SOVD has the following key properties:

- Central edge node communication
- Usage of state-of-the-art technologies
- Support of remote, proximity and in-vehicle diagnostics use-cases
- Self-explaining protocol (no need for ODX based interpretation)
- UDS as a subset
- Support for HPC use-cases

2 SOVD Reference Architecture

Providing a central SOVD edge node in a distributed system does require some infrastructural components. To achieve this a reference architecture that clusters functions into several blocks has been introduced. This architecture is displayed in Figure 2.1).

Main components of this reference architecture are:

- SOVD Gateway
- Diagnostic Manager
- SOVD to UDS Translation

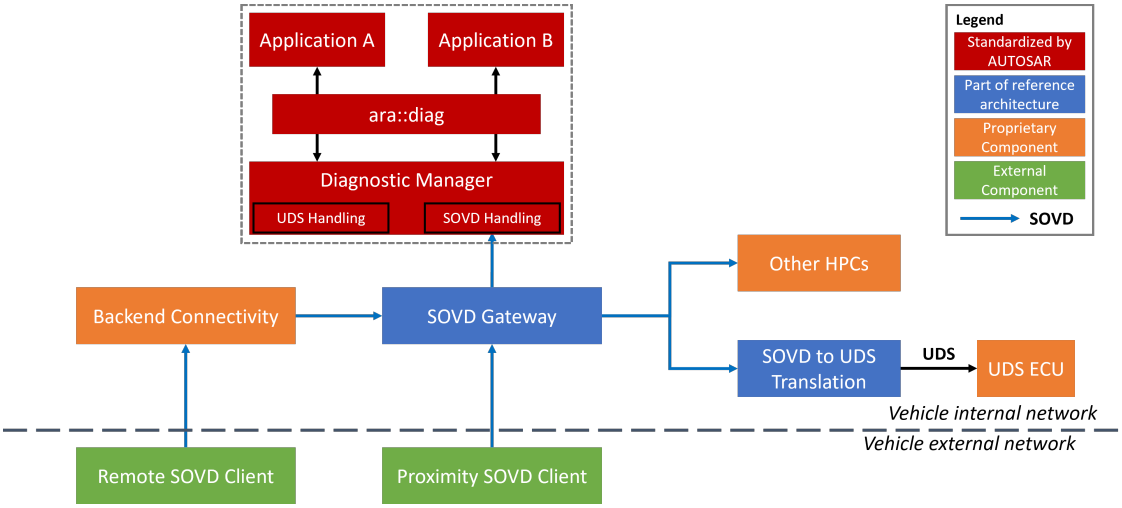

Figure 2.1: SOVD Reference architecture

2.1 SOVD Gateway

Upon request from a SOVD client the SOVD Gateway acts as SOVD edge node and routes the requests to respective internal SOVD endpoints. The routing takes place based on the entity part in the URI of the SOVD request. The SOVD Gateway must extract this part of the URI and route towards the corresponding endpoint. The setup of potential internal SOVD endpoints can be statically configured or dynamically discovered using mDNS. The forwarding itself takes place on application layer, i. e. the SOVD gateway acts as HTTP reverse proxy.

For the configuration of the SOVD Gateway the SOVDGatewayInstantiation has been introduced in the TPS_Manifest. This manifest allows to configure the external SOVD connection towards SOVD clients as well as the internal forwarding targets.

2.2 Diagnostic Manager

Before the introduction of SOVD the Diagnostic Manager's main purpose was to handle diagnostic services and fault memory according to ISO 14229-1. With the introduction of SOVD the Diagnostic Manager also aims on natively supporting SOVD and therefore acts as an SOVD server. One of the main guiding principles for the introduction of SOVD was to reuse as much of existing functionality of UDS as suitable while not limiting the native support of SOVD. On structural level the Diagnostic manager does allow multiple Diagnostic Server Instances. This aims on keeping individual Software Clusters independent. Each Diagnostic Contribution Set in the DEXT represents one Diagnostic Server Instance with an individual Diagnostic Address. This addressing principle was adapted by SOVD. The Diagnostic Manager itself represents a SOVD component, whereas each Diagnostic Server Instance is represented by an SOVD subcomponent. Still the Diagnostic Manager acts as an SOVD server and routing towards each subcomponent is handled within the Diagnostic Manager. For the configuration of this SOVD server the SOVDServerInstance has been introduced in the TPS_Manifest. The internal behavior of the Diagnostic Manager does depend on the SOVD use case and will be discussed in detail in SWS_Diagnostics.

2.3 SOVD to UDS Translation

This functional block allows the translation of SOVD commands to UDS requests, based on predefined ODX definition of SOVD to UDS mapping. The details of this SOVD to UDS translation are defined by ASAM. This functional block shall be treated as an on-board test client and will send UDS requests to the target diagnostic address and send translated UDS responses to SOVD clients. The complexity of this functional block from implementation point of view is high. But since ASAM has defined these details, AUTOSAR will just refer to ASAM standard at this point.

2.4 Backend Connectivity

SOVD aims on supporting proximity, remote and in-vehicle diagnostics. SOVD has standardized proximity and in-vehicle access rather precisely by introducing mDNS. Standardizing remote access is rather difficult since many dependencies towards the backend infrastructure exist. This degree of freedom is kept open by AUTOSAR. Nonetheless a rather straightforward solution is possible by abstracting the backend connection by some backend connectivity functional block, which routes the SOVD requests towards the SOVD Gateway. Discovery of the SOVD Gateway by the Backend Connectivity function block can be accomplished by using mDNS, while routing can be realized by HTTP forwarding.

3 SOVD use cases

This chapter describes how SOVD use cases are designed and achieved on level of a Diagnostic Server Instance.

3.1 Common use cases in SOVD and UDS

A big portion of the SOVD use cases can be mapped in a straightforward fashion to existing UDS use cases. How this mapping has been done in the context of the Diagnostic manager is displayed in SWS Diagnostics. For cases matching UDS (ISO 14229-1) one of the main principles was introduced: Reuse of the same port instance as used for the corresponding UDS service. This is especially convenient for designing applications since no further ports need to be integrated with redundant functionality. Also, for these kinds of implementations the same rules regarding reentrancy and parallel execution as for UDS apply. SOVD handling of some use-cases may be sufficiently different from their UDS counterparts requiring differentiated requirements and mechanisms. Also, in terms of methodology these SOVD methods are adopting the existing configuration mechanism by using DEXT and only extending the DEXT at certain needed places. This matching has been done in the following way:

- SOVD data is derived from DiagnosticDataIdentifier
- SOVD configuration is derived from DiagnosticDataIdentifier
- SOVD operation is derived from DiagnosticRoutine
- SOVD fault is derived from DiagnosticTroubleCodeUDS

Besides these configurable methods also two standardized modes have been introduced which match the existing ISO 14229-1 services 0x28 CommunicationControl and 0x85 ControlDTCSettings. The static meta data needed for SOVD are derived from semantically corresponding DEXT and Manifest attributes. Details of this mapping are listed in the SWS_Diagnostics and in the TPS_Manifest. For the dynamic data which is passed from application to the Diagnostic Manager an interpretation must be done by the Diagnostic Manager. This is a big contradiction to the UDS (ISO 14229-1) solution, where the Diagnostic Manager could simply pass the data towards the tester, which in return was parameterized. These dynamic data are expressed in the DEXT by DiagnosticDataElements. For expressing the data interpretation the compuMethod within the SwDataDefProps of the DiagnosticDataElement is used. The Diagnostic Manager must then use this compuMethods to convert the byte stream from the application to valid SOVD JSON for the SOVD client and vice versa. Details on how this conversion shall be done is expressed in SWS_Diagnostics.

3.2 SOVD-specific use cases

Use cases that are exclusively for SOVD and find no match in ISO 14229-1 are also supported by the Diagnostic Manager. For these use cases new interfaces are introduced to the Diagnostic Manager that are exclusively used by SOVD. Currently two new interfaces have been introduced for Authorization and Proximity Challenge. Additionally, the SOVD use case data-lists is handled by the Diagnostic Manager natively. Further SOVD specific use cases will be added in further releases.

3.2.1 Access Permissions

Authorization in SOVD is handled using OAuth tokens that are part of the request header. When a request with OAuth token is received, the Diagnostic Manager uses the interface SovdAuthorization to request the encoded authorization role from the application. With this authorization role the Diagnostic Manager can determine whether the SOVD Client is authorized to perform the requested SOVD. The existing DEXT model for Service 0x29 of ISO 14229-1 is utilized for the underlying roles, including the DiagnosticAuthRoles and the relationship between DiagnosticAccessPermission and the individual SOVD methods. This mapping is based on the existing services described in the SOVD section of the Diagnostic Manager, ensuring the proper association between DiagnosticAccessPermission and the corresponding SOVD methods.

3.2.1.1 Proximity Challenge

SOVD proximity challenge is used by the SOVD server to challenge the client for prove of proximity before executing a SOVD operation. For this challenge, the new interface SovdProximityChallenge was introduced that hands the challenge procedure to the application and informs the Diagnostic Manager about the result. Based on the result the Diagnostic Manager will then perform the operation. The configuration whether a SOVD operation does require a proximity challenge or not is also done by DEXT using a special data group (SDG). For details of this SDG see SWS_Diagnostics

3.2.2 Software Update

The Software and Configuration Update functionality is realized through using the AUTOSAR Update and Configuration Management (UCM) framework. There are two ways to perform Software Updates using UCM:

- Coordinated vehicle update using UCM campaign (through UCM VUCM and Vehicle Package), a.k.a. "vehicle update"
- Individual updates of specific components on the vehicle (through UCM Diagnostic Application running on Adaptive Machines), a.k.a "component update"

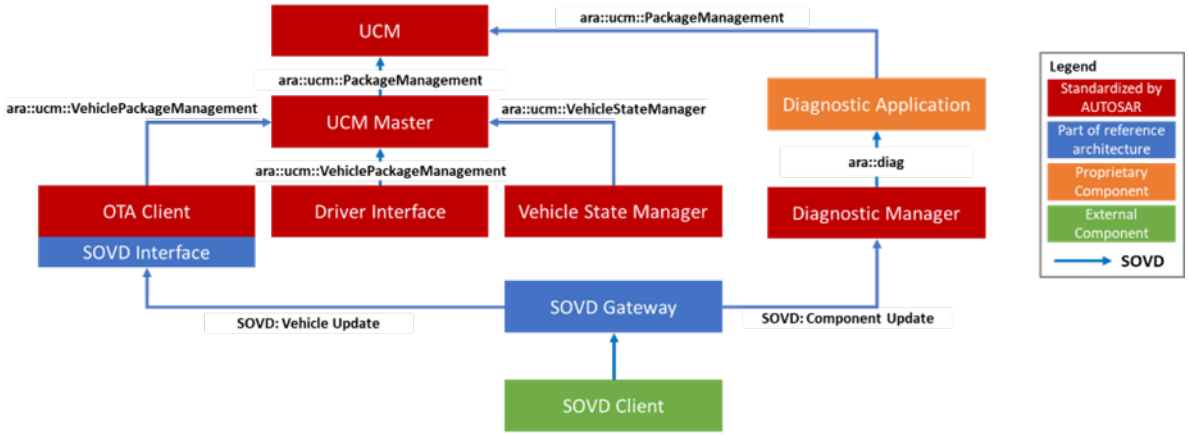

Figure 3.1: Solution Approach for SOVD updates

In the case of a Vehicle Update the SOVD Gateway routes the request initiated by any SOVD Client towards an SOVD interface of an OTA client, which handles the updates independently, without involving the Diagnostic Manager. As mentioned in "6.10 API Methods for Software Update" in ASAM SOVD, API Specification Version 1.0.0 documentation, the automated software update puts the SOVD server in control of the complete update process for the update package selected by the client. It means that OTA Client acts as SOVD Server responsible for receiving a single SOVD REST automated software update request and will drive the vehicle update process by using Vehicle/-Software Packages provided by backend and rerouting them to VUCM instance using ara::vucm VehiclePackageManagement Service Interface. Implementation details of this behaviour are platform vendor specific and, since the standardization of the OTA client is out of scope of AUTOSAR UCM specification, are left out of the scope of this document as well.

However, in the case of component update, the Diagnostic Manager plays a crucial role. The Diagnostic Manager provides a standardized interface, ara::diag, which is mapped to the ara::ucm::PackageManagement API through UCM Diagnostic Application. The Diagnostic Manager can control the forwarding of request messages related to software updates. These functions allow the registration and deregistration of the update service, respectively. The integration between ara::diag and ara::ucm ensures seamless handling of component updates in the AUTOSAR Adaptive Platform.

In contrast to "vehicle update", this approach requires a high degree of the involvement from SOVD Client as it needs to drive the specific component update through different stages of UCM update, including downloading software packages to the vehicle, transferring them to a target Machine, processing, activating them and deleting the resources

3.2.3 Logging

The logging functionality within the system is not standardized by ara::log, and as a result, the specifics of logging file retrieval and management are handled in a plat-

form vendor-specific manner. This means that the connection between ara::diag and ara::log, particularly for reading out logging files, is implemented based on the platform vendor's guidelines and requirements.

3.2.4 Bulk Data

The Diagnostic Manager, through its ara::diag interface, is responsible for managing the various aspects of bulk data operations. These operations are facilitated by the Adaptive application, which can utilize persistence mechanisms and cryptographic functionalities such as signature generation and hash verification. By leveraging these capabilities, the Adaptive application can handle the storage, retrieval, and deletion of bulk data in a secure and controlled manner.

3.2.5 Configuration

The handling of configurations as understood by ASAM involves two methods: Reading and writing configurations as parameters, and reading and writing configurations as bulk-data. Support for configurations as bulk-data is achieved through the introduction of a new interface called ara::diag::GenericConfiguration. This interface encompasses all the methods defined in the ASAM SOVD standard. The ara::diag::GenericConfiguration interface enables the Diagnostic Manager to perform multiple operations, based on the granted access control. These operations can be carried out by Adaptive applications using persistence and cryptographic operations. By leveraging the ara::diag::GenericConfiguration interface, the handling of configurations as bulk-data is streamlined, conforming to the ASAM SOVD standard and providing flexibility for configuration management in automotive systems.

4 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant for this document:

Abbreviation / Acronym:	Description:
AA	AUTOSAR Adaptive Application
AP	AUTOSAR Adaptive Platform
API	Application Programming Interface
ASAM	Association for Standardization of Automation and Measuring Systems
BSW	Basic Software
DEXT	AUTOSAR Diagnostic Extract[2], describing diagnostic configuration of an ECU
DID	Data Identifier according to ISO 14229-1[3]. This 16 bit value uniquely defines one ore more data elements (parameters) that can are used in diagnostics to read, write or control data.
DM	AUTOSAR Adaptive Diagnostic Management
DNS	Domain Name System
DoIP	Diagnostics over Internet Protocol (Communication protocol of automotive electronics according to ISO-13400-2[4])
DTC	Diagnostic Trouble Code according to ISO 14229-1[3]
ECU	Electronic control unit
EDR	Extended Data Record
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JSON	JavaScript Object Notation
MetaInfo	Meta-Information in the form of a key-value map, which is given from DM to external service processors.
NRC	Negative Response Code used by UDS in the diagnostic response to indicate the tester that a certain failure has occurred and the diagnostic request was not processed.
ODX	Open Diagnostic Data Exchange (Standardized diagnostic data description format as of ISO-22901)
REST	Representational State Transfer
SDG	Special Data Group
SID	Service Identifier, identifying a diagnostic service according to UDS, such as 0x14 ClearDiagnosticInformation
SOVD	Service-Oriented Vehicle Diagnostics
TLS	Transport Layer Security
UDS	Unified Diagnostic Services
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Terms:	Description:
Diagnostic Server instance	Diagnostic Server (DM) is intended to support an own Diagnostic Server instance per installed *SoftwareCluster*, see section 2.2 for a detailed description. Each of those Server instances has and manages its own resources and is responsible for dispatching and processing of diagnostic services.

Terms:	Description:
Extended Data Records	Contains statistical data for a DTC. Extended data records are assigned to DTCs and maintained and stored by the DM.
Snapshot Record	Snapshots (sometimes referred to as freeze frames) are specific data records associated with a DTC and stored in the fault memory at a certain point of time during fault detection. The DTC specific data-parameters are intended to ease the fault isolation process.
SoftwareCluster	A SoftwareCluster groups all AUTOSAR artifacts which are relevant to deploy software on a machine. This includes the definition of applications, i.e. their executables, application manifests, communication and diagnostics. In the context of diagnostics a SoftwareCluster can be addressed individually by its own set of diagnostic addresses.
SOVD entity	Entity in context of SOVD[1]
SOVD lock	SOVD locking mechanism as defined by [1]
UDS service	A diagnostic service as defined in ISO 14229-1[3].
Non-volatile Memory	In the context of DM, Non-volatile Memory refers to the persistent information over the shutdown of the DM process. This does not depend on HW details.

5 Related documentation

5.1 Input documents & related standards and norms

[1] ASAM SOVD Service-Oriented Vehicle Diagnostics - API Specification V1.0.0
http://www.asam.net

[2] Diagnostic Extract Template
AUTOSAR_CP_TPS_DiagnosticExtractTemplate

[3] ISO 14229-1(2020) – Unified diagnostic services (UDS) – Part 1: Application layer (Release 2020-02)
https://www.iso.org

[4] ISO 13400-2:2019 – Road vehicles – Diagnostic communication over Internet Protocol (DoIP) – Part 2: Network and transport layer requirements and services (Edition 2, Release 2019-12)
https://www.iso.org/standard/74785.html