ETSI TS 103 882 V2.1.1 (2024-05)

Intelligent Transport Systems (ITS);
Automated Vehicle Marshalling (AVM);
Release 2

Reference
DTS/ITS-001958

Keywords
ITS, service

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Content

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Intelligent Transport Systems (ITS).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Automated Vehicle Marshalling (AVM) in the context of road traffic, means that individual or multiple unoccupied vehicles of several kinds are automated driven in the lower velocity range. The AVM entity in the Facilities layer is a broad Vehicle Motion Control (VMC) functionality supporting Remote vehicle Operation (RO) in the low-speed domain and it is an entity within the Intelligent Transport Systems (ITS) ecosystem. Some of the use cases supported by the AVM entity are:

- Automated Valet Parking Systems (AVP) [i.6] and electric charging:
  - A feature to provide automated parking and electric charging services.
- Automated Vehicle Marshalling in vehicle factories:
  - A logistic feature to improve the efficiency of production and vehicle transport within vehicle factories.

- Automated Vehicle Marshalling in depots:
    - A logistic feature for shunting goods inside a depot, improve traffic and safety, and provide additional logistic efficiencies.

Such use cases are summarized under the term AVM. Typical vehicles for these applications are passenger cars, delivery vans and trucks. It does not mandatorily mean for the system that the vehicles are moved driverless, however in most use cases the operations are planned without the need of driver's supervision thus operations are categorized as Level 4 - High Driving Automation according to the Surface Vehicle Recommended Practice SAE J3016 [i.1].

The remote Vehicle Motion Control communication protocol described in the present document focuses on an infrastructure-lead implementation of an AVM system analogue to a Type 2 AVP system implementation as described in ISO 23374-1 [i.6]. In this type of marshalling systems, a Subject Vehicle (SV) is controlled by a smart infrastructure that observes them by external sensors mounted in direct vicinity of the driving path.

The operation interface supporting AVM systems is an implementation of the remote VMC communication protocol as a regular exchange of information between SVs and the RO sub-system on a wireless network and therefore it is a Vehicle-to-Everything (V2X) communication. The present document defines the V2X communication messages supporting the implementation of the operation interface for AVM applications.

1 Scope

The present document specifies Automated Vehicle Marshalling (AVM) messages at the Facilities layer, which are agnostic to the lower layers and thus supported by different transport and network layer protocols e.g. Basic Transport Protocol (BTP)/Geonetworking, or Internet Protocol (IP) based stacks.

AVM is a specific category of low-speed remote controlled automated driving. The messages and protocol for the Automated Vehicle Marshalling entity at the Facilities layer defined in the present document are applicable to other use cases in the category of low speed controlled automated driving.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] Recommendation ITU-T X.680 (May 2021): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] SAE J3016 (April 2021): "Surface Vehicle Recommended Practice - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles".

[i.2] Autosar ID 849, R22-11: "Autosar Standard: Foundation - E2E Protocol Specification".

[i.3] Autosar ID 016, R21-11: "Autosar Standard: Classic Platform - Specification of CRC Routines".

[i.4] Recommendation ITU-T X.691: "Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)".

[i.5] IEEE 1609.2™-2022: "Standard for Wireless Access in Vehicular Environments--Security Services for Application and Management Messages".

[i.6] ISO 23374-1 (July 2023): "Intelligent transport systems -- Automated valet parking systems (AVPS) - Part 1: System framework, requirements for automated driving, and communication interface".

[i.7] VDA Position: "Automated Valet Parking Systems Requirements for Automated Valet Parking Systems".

[i.8] 5GAA T-20002 (May 2023): "Automated Valet Parking Technology Assessment and Use Case Implementation Description".

[i.9] ISO 26262 series: "Road vehicles - Functional Safety".

[i.10] IEC 61508-1 (April 2010): "Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements".

[i.11] ETSI TS 102 894-2 (V2.1.1): "Intelligent Transport Systems (ITS); Users and applications requirements; Part 2: Applications and facilities layer common data dictionary; Release 2".

[i.12] ETSI TS 103 898 (V2.0.0): "Intelligent Transport Systems (ITS); Communications Architecture; Release 2".

[i.13] ETSI TS 102 940 (V2.1.1): "Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management; Release 2".

[i.14] ETSI TS 103 141 (V2.2.1): "Intelligent Transport Systems (ITS); Facilities layer function; Multi-Channel Operation (MCO) for Cooperative ITS (C-ITS); Release 2".

[i.15] ETSI TS 102 723-5 (V2.0.0): "Intelligent Transport Systems (ITS); OSI cross-layer topics; Part 5: Interface between management entity and facilities layer; Release 2".

[i.16] ETSI TS 103 097 (V2.1.1): "Intelligent Transport Systems (ITS); Security; Security header and certificate formats; Release 2".

[i.17] Recommendation ITU-T X.509: "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the term **authority** given in ISO 23374-1 [i.6] and the following apply:

Automated Vehicle Marshalling (AVM): orchestration and remote VMC of vehicles at low speed

communication protocol: formal description of a message format and its rules

logical architecture: decomposition of a system into functional components without considering their deployment

Subject Vehicle (SV): vehicle supporting the AVM entity

System Operator (SO): entity that receives or hands over authority of an SV from or to the user and manages AVM operation

user: beneficiary in an AVM system capable to hand over authority to an SV or retrieve authority from an SV

Vehicle-to-Everything (V2X) communication: communication either Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I) and/or Infrastructure-to-Vehicle (I2V)

Vehicle Motion Control (VMC): control of the vehicle motion, that is the vehicle translation and rotation around all three axes in a Cartesian coordinate system

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

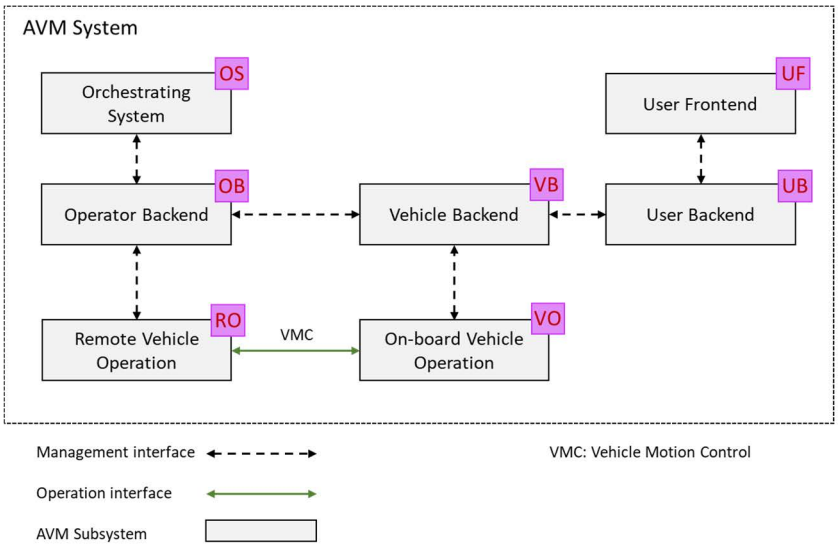
5GAA	5G Automotive Association
ADU	Application Data Unit
ASIL	Automotive Safety Integrity Level
ASN.1	Abstract Syntax Notation One
AVM	Automated Vehicle Marshalling
AVMS	Automated Vehicle Marshalling System
AVP	Automated Valet Parking
BTP	Basic Transport Protocol
CA	Certificate Authority
CDD	Common Data Dictionary
C-ITS	Cooperative Intelligent Transport System
CRC	Cyclic Redundant Check
DCC	Decentralized Congestion Control
DDP	Device Data Provider
DE	Data Element
DF	Data Frame
DTLS	Datagram Transport Layer Security
E2E	End-to-End
FL-SDU	Facility-layer Service Data Unit
ID	Identifier
IEC	International Electrotechnical Commission
IP	Internet Protocol
IT	Information Technology
ITS	Intelligent Transport System
ITS-S	Intelligent Transport System-Station
LDM	Local Dynamic Map
MCO	Multi-Channel Operation
MIB	Management Information Base
MIM	Marshalling Infrastructure Message
MTU	Maximum Transmission Unit
MVM	Marshalling Vehicle Message
OB	Operator Backend
OBU	On-Board Unit
OEM	Original Equipment Manufacturer
OID	Object Identifier
OS	Orchestrating System
PCI	Protocol Control Information
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
POTI	Position and Time
RO	Remote vehicle Operation
RSU	Road Side Unit
SA	Service Announcement
SHA	Secure Hash Algorithm
STD	State Transition Diagram
SV	Subject Vehicle
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TR	Technical Report
UB	User Backend
UDP	User Datagram Protocol
UF	User Frontend
UPER	Unaligned Packed Encoding Rules

URL	Uniform Resource Locator
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
VB	Vehicle Backend
VDA	Verband der Automobilindustrie
VIN	Vehicle Identification Number
VMC	Vehicle Motion Control
VO	on-board Vehicle Operation
XER	XML Encoding Rules
XML	eXtensible Markup Language
XOR	eXclusive OR

4 AVM entity introduction and general functionality

4.1 Introduction

The AVM entity is a functionality supporting marshalling operations in an AVM system. An illustrative logical architecture of an AVM system is given in Figure 1. The AVM entity relates to an implementation of the operation interface Vehicle Motion Control (VMC) shown in green in Figure 1.



Subsystem	Description
OS	Orchestrating System
OB	Operator Backend
RO	Remote Vehicle Operation
VB	Vehicle Backend
VO	On-board Vehicle Operation
UB	User Backend
UF	User Frontend

Figure 1: Illustrative logical architecture of an AVM system

The present clause is organized as follows:

- Clause 4.2 introduces the AVM entity at the Facilities layer.
- Clause 4.3 describes the general functionality of the AVM entity.

4.2 AVM entity at the Facilities layer

Within the scope of the present document, the term AVM entity as shown in Figure 2, refers to a Facilities layer implementation of the AVM messages Marshalling Infrastructure Message (MIM) and Marshalling Vehicle Message (MVM). The term "message" refers to the Facilities layer Protocol Data Unit (PDU) and the term "payload" refers to the applications layer Application Data Unit (ADU).

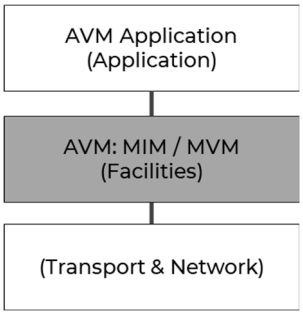


Figure 2: AVM entity at the Facilities layer

The AVM entity supports the management and dissemination of the AVM messages as bidirectional communication between the Remote Vehicle Operation (RO) subsystem of the AVM service provider and the On-board Vehicle Operation (VO) subsystem in the SV as depicted in Figure 1.

Within the scope of the present document this functionality corresponds to the definition and specification of the following AVM message types as part of the AVM entity:

- MIM as specified in clause 7 and in ASN.1 notation in Annex A. The corresponding entity is referred as "Marshalling Infrastructure Message".
- MVM as specified in clause 8 and in ASN.1 notation in Annex A. The corresponding entity is referred as "Marshalling Vehicle Message".

4.3 General functionality

4.3.1 Introduction

The AVM entity enables the marshalling operation described in clause 4.1 and therefore it has a clear boundary and functionality within an AVM architecture. This clause defines the boundary and explains the functionality of the AVM entity as follows:

- Clause 4.3.2 defines the functional boundary of the AVM entity.
- Clause 4.3.3 explains the functionality that the AVM entity enables.
- Clause 4.3.4 explains how the AVM entity supports automated vehicle operation.

4.3.2 AVM entity functional boundary

The AVM entity supports AVM operation when the SV is under system Authority, in other words, the System Operator (SO) has authority to perform marshalling operations at the SV through the operation interface VMC stablished between the RO and the VO subsystems. The functional boundary of the AVM entity is shown in Figure 3. A definition of the states shown in Figure 3 is given in ISO 23374-1 [i.6].

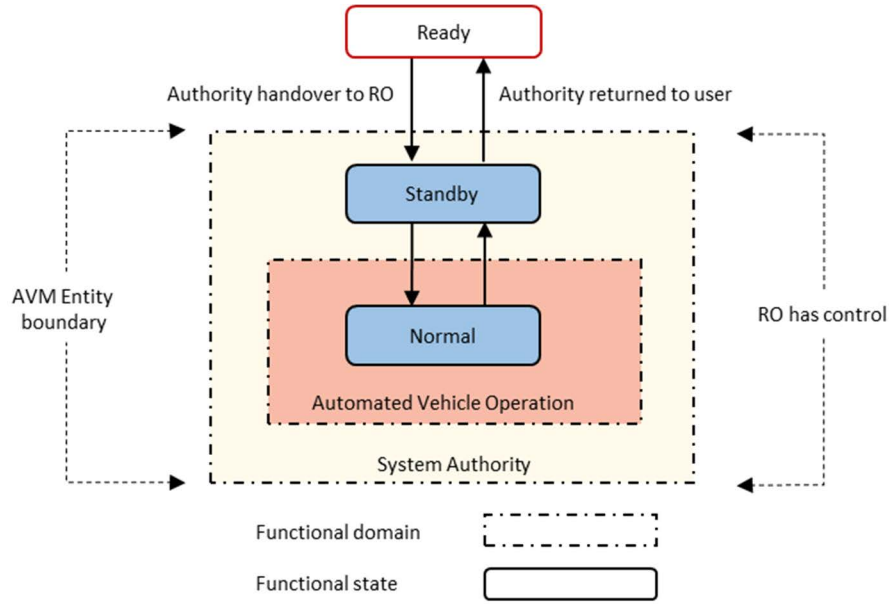


Figure 3: Simplified state transition diagram showing the operation boundaries of the AVM entity

4.3.3 AVM operation interface

Within an AVM architecture, such as the one shown in Figure 1, the AVM operation is enabled by an implementation of the VMC operation interface between the RO and the VO subsystems as shown in Figure 4. The communication between the AVM entity in the RO system and the AVM entity in the VO system is supported by the AVM messages at the Facilities layer defined in clause 4.2. The AVM messages used by the AVM entity are from the communication link point of view technology agnostic, thus allowing an implementation of the VMC operation interface through diverse communication technologies.

AVM operation refers to the action of safely manoeuvring the SV in an AVM facility.

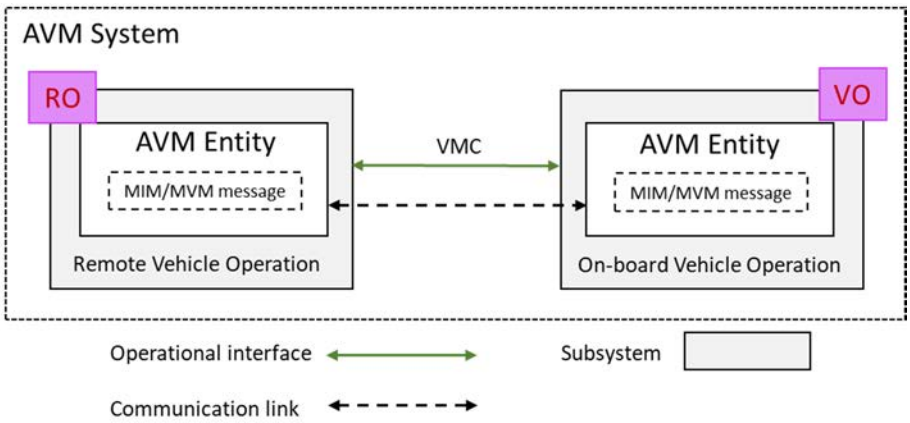


Figure 4: Vehicle Motion Control operation interface

4.3.4 AVM Automated Vehicle Operation

The AVM entity supports two methodologies to realize the VMC operation interface within the Automated Vehicle Operation domain shown in Figure 3:

1) The **Path Control** methodology, which is based on the transmission of way points from the RO to the VO. Through an implementation of the VMC operation interface using the Path Control methodology, the SV receives the geometry of the desired driving path, and it is responsible to follow it by its own internal control algorithms. The Data Elements (DE) particular to this methodology are described in clause 7.6. Some implementation considerations are given in the informative clause E.6 and the methodology is described in detail in [i.7].

2) The **Trajectory Control** methodology, which is based on the transmission of control points from the RO to the VO. Through an implementation of the VMC operation interface using the Trajectory Control methodology, the SV receives control commands to fulfil a planned trajectory. The DE related to this methodology are described in clause 7.7 with some implementation considerations described in informative clause E.7.

   The Trajectory Control methodology allows choosing between controlling the longitudinal acceleration or the longitudinal velocity of the SV. Longitudinal acceleration and longitudinal velocity are motion parameters:

   - Trajectory Control based on acceleration: longitudinal acceleration is the motion parameter as described in clauses 7.7.5.3 and 7.7.5.4.

   - Trajectory Control based on velocity: longitudinal velocity is the motion parameter as described in clauses 7.7.5.3 and 7.7.5.5.

The OS uses only one methodology for AVM operation of a particular SV. For both methodologies, the selection of an appropriate number of way points or control points may depend on the state of the communication channel e.g., through data provided by lower layers or Multi-Channel Operation (MCO) - ETSI TS 103 141 [i.14].

5 Architecture and interface description

5.1 Introduction

The AVM entity features several functionalities which are specified hereafter. The present clause is organized as follows:

- Clause 5.2 introduces the AVM entity architecture.
- Clause 5.3 defines functionalities of the AVM entity.
- Clause 5.4 defines the interfaces for the AVM entity.

5.2 AVM entity architecture in the ITS ecosystem

The AVM entity is a Facilities layer entity in the ITS-S architecture as defined in ETSI TS 103 898 [i.12]. It may interface with other entities of the Facilities layer and with ITS applications (in particular, with the AVM application). Figure 5 depicts the AVM entity within the ITS-S architecture along with the logical interfaces to other layers and entities within the Facilities layer.

The entities for the collection of data to generate AVM messages within the AVM entity (i.e. MIMs or MVMs) may be the Device Data Provider (DDP) and the Position and Time management (POTI). For vehicle ITS-Ss, the DDP may be connected to the in-vehicle network to provide the vehicle state information. For roadside ITS-Ss and central ITS-Ss, the DDP may be connected to sensors mounted on the roadside infrastructure. The POTI entity provides estimates of the kinematic state of the ITS-S and time information. The AVM entity may also interface with other entities in the Facilities Layer such as the Service Announcement (SA) entity to indicate an ITS-S's ability to generate AVM messages and to provide details about the communication technology used or the MCO entity for the purpose of generation, transmission and reception of AVM messages.

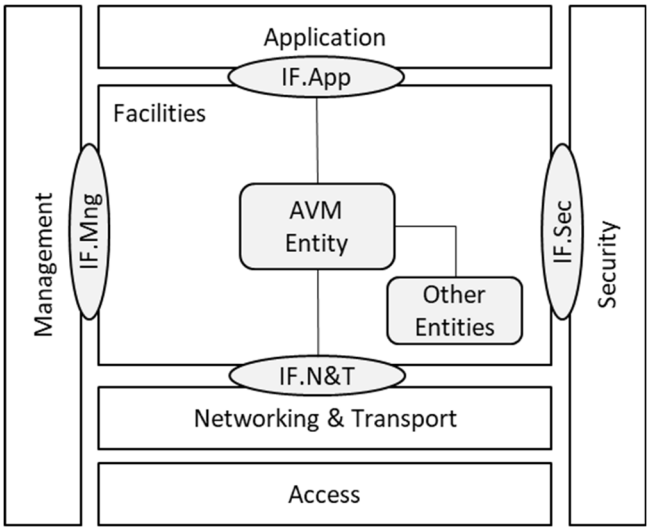


Figure 5: AVM service within the ITS-S architecture

5.3 AVM entity functional architecture

The AVM entity shall provide at least the following functions in the RO subsystem and the VO subsystem for sending and receiving the AVM messages defined in clause 4.2:

- Encode AVM message: This sub-function shall construct the AVM message according to the format specified in normative Annex A and informative Annex B.
- Decode AVM message: This sub-function shall decode the received AVM message.
- AVM message transmission management: This sub-function implements the protocol of the originating ITS-S including:
  - Activation and termination of AVM message transmission operation.
  - Determination of the AVM message generation event frequency.
  - Triggering the generation of the AVM message.
- AVM message reception management: This sub-function implements the protocol in the receiving ITS-S including:
  - Triggering the decoding of the AVM message upon receiving an incoming AVM message.
  - Provisioning of the decoded AVM message to the Local Dynamic Map (LDM) and/or ITS applications of the receiving ITS-S.
  - Optionally, checking the validity of the information of received AVM messages.

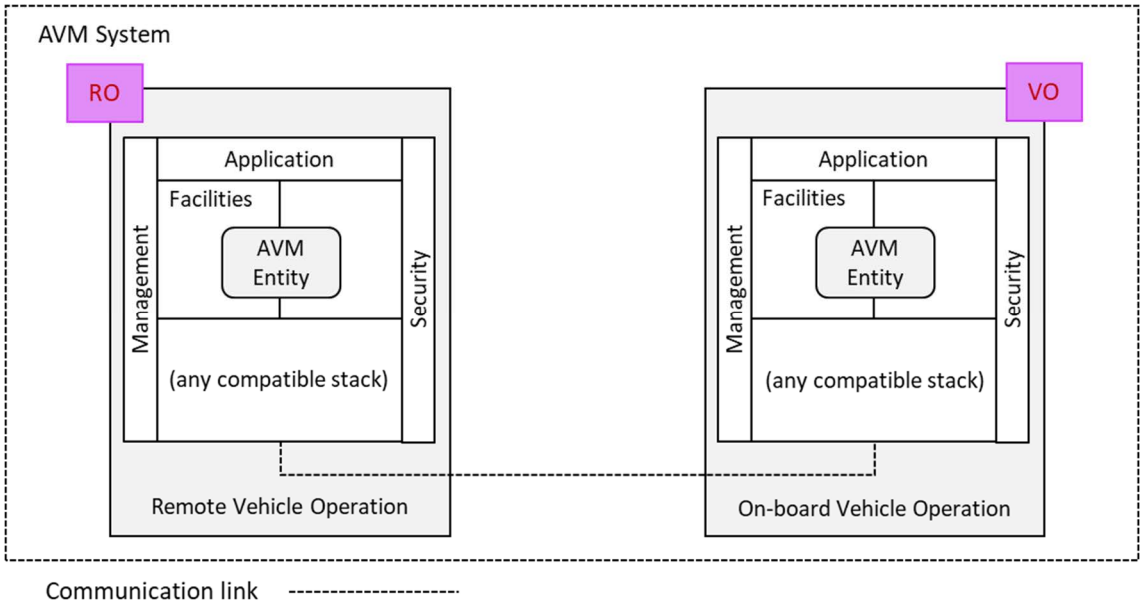


Figure 6: AVM entity functional architecture within an AVM system

5.4 Interfaces to the AVM entity

5.4.1 Introduction

The AVM entity may interface to other entities and communication layers for the realization of the functionality specified in clauses 5.2 and 5.3. This clause presents such interfaces and is organized as follows:

- Clause 5.4.2 explains interfacing to ITS applications.
- Clause 5.4.3 explains interfacing to other Facilities layer entities.
- Clause 5.4.4 describes the interface to the Network & Transport layer.
- Clause 5.4.5 explains interfacing to the ITS Management entity.
- Clause 5.4.6 explains interfacing to the Security entity.

5.4.2 Interface to ITS applications

An ITS application, such as the AVM application, is an application layer functionality that implements the logic for fulfilling one or more ITS use cases. For the provision of received data, the AVM entity provides the interface IF.App (also implemented as IF.AVM) to ITS application as illustrated in Figure 5.

5.4.3 Interface to other Facilities layer entities

For the generation of MIMs and MVMs, the AVM entity may interact with other Facilities layer entities to obtain the required data. This set of other facilities is referred to as data provisioning facilities, e.g. the ITS-S's POTI or DDP mentioned in clause 5.2. Data is exchanged between the data provisioning facilities and the AVM-S via the interface IF.FAC.

For the transmission and reception of MIMs and MVMs, the AVM entity may interact with other Facilities layer entities via the interface IF.FAC.

If the ITS-S supports MCO, the AVM entity exchanges information via the interface IF.MCO. This interface is specified via MCO_FAC in ETSI TS 103 141 [i.14]. The exchanged information can be used by the AVM entity to adjust the message generation frequency via the parameters *T_GenMIM* and *T_GenMVM* described in clause 6.2, or the message size via the number of way points (clause 7.6.2) or control points (clause 7.7.4) based on the resources available on the used communication channel.

5.4.4 Interface to Network & Transport layer

The AVM entity can use any compatible communication stack for the transmission and reception of MIMs and MVMs as illustrated in Figure 6.

At the originating ITS-S, the AVM entity can provide the MIM or MVM embedded in a Facility-Layer Service Data Unit (FL-SDU) to the ITS Network & Transport layer, optionally together with Protocol Control Information (PCI). At the receiving ITS-S, the ITS Network & Transport Layer passes the received AVM message to the AVM entity.

The data set that can be passed between the AVM entity and the ITS Networking & Transport Layer for the originating and receiving ITS-S are specified in Table 1.

Table 1: Data passed between the AVM entity and the ITS Network & Transport Layer

Category	Data	Data requirement	Mandatory/Optional
Data passed from the AVM entity to the ITS Networking & Transport Layer	MIM or MVM	*{MIM}* or *{MVM}* as specified in Annex A	Mandatory
	PCI	Depending on the protocol stack applied in the networking and transport layer and out of scope of the present document	Optional
Data passed from the ITS Networking & Transport Layer to the AVM entity	Received MIM or MVM	*{MIM}* or *{MVM}* as specified in Annex A	Mandatory

5.4.5 Interface to the ITS management entity

The AVM entity of an originating ITS-S obtains configuration parameters and other control information from the Management Information Base (MIB) located in the Management entity via the IF.Mng interface, as shown in Figure 5.

NOTE 1: A list of primitives exchanged with the management layer are provided in ETSI TS 102 723-5 [i.15].

NOTE 2: Specifications of IF.Mng and the corresponding protocol are out of scope of the present document.

5.4.6 Interface to the security entity

The AVM entity may exchange primitives with the Security entity of the ITS-S using the IF.Sec interface provided by the Security entity as depicted in Figure 5.

NOTE: Depending on the security requirements of AVM implementation scenarios, different security solutions may be used. Specification of the interface configuration related to a specific security solution entity is out of scope of the present document.

6 MIM and MVM dissemination

6.1 Introduction

This clause introduces requirements and constraints for the dissemination of MIM and MVM messages and it is structured as follows:

- Clause 6.2 specifies requirements for the generation of MIMs and MVMs.
- Clause 6.3 defines constraints for the dissemination of MIMs and MVMs.

6.2 MIM and MVM generation

MIM messages shall be generated upon the activation of the AVM entity.

MVMs shall only be generated by SV ITS-Ss and MIMs shall only be generated by SO ITS-Ss. The generation of MVMs by SV ITS-Ss shall start after the first successful reception of a MIM with a valid addressing to the SV ITS-S receiving the MIM.

Each MIM shall include one or more `Mim` container. Each `Mim` container shall address one and only one SV ITS-S.

A new MIM message shall be generated if the time elapsed since the last MIM generated with a `Mim` container addressing any of the SV ITS-S under control is equal or greater than *T_GenMIM*. The generated MIM shall include all Mim containers generated longer than *T_GenMIM* ago.

A MIM message can also be generated under certain conditions or events, e.g. a driving permission change or triggering an emergency stop. In any case, the number of MIMs generated by an RO ITS-S during a time interval *T_EventMIM* with a message interval less than *T_GenMIM* shall be less or equal than *N_EventMIM*.

The generation of MVMs by the SV ITS-Ss shall start after the first successful reception of a MIM with a valid addressing to the SV receiving the MIM. AVM application instances in ITS stations in SVs determine whether or not to process a MIM based on the contents of the *SessionID* or *MissionID* fields (or both). These fields are defined in ISO 23374-1 [i.6].

A new MVM shall be generated if the time elapsed since the last MVM generated equal or greater than *T_GenMVM*. A MVM can also be generated if certain events occur (e.g. to trigger an immediate *TimeSyncResponse*). In any case, the number of MVMs generated by an SV ITS-S during a time interval *T_EventMVM* with a message interval less than *T_GenMVM* shall be less or equal than *N_EventMVM*.

Informative clause C.4 declares recommended values for the generation parameters.

NOTE: Different implementations of the AVM entity are possible based on the AVM functional architecture and the use case deployed.

6.3 MIM and MVM dissemination constraints

In an implementation of the AVM entity, some constraints for the dissemination of MIMs and MVMs related to the Maximum Transmission Unit (MTU) size limit of the Network & Transport layer exist. The size of an ASN.1 encoded MIM or MVM shall not exceed *MTU_MIM* or *MTU_MVM*, respectively. *MTU_MIM* and *MTU_MVM* depend on the MTU of the Access Layer technology (*MTU_AL*) over which the message is transported. If multiple channels are available and, e.g. the ITS-S supports MCO, these restrictions shall apply to each channel individually. *MTU_MIM* and *MTU_MVM* shall be less than or equal to *MTU_AL* reduced by the header size of the Facilities layer protocol and the header size of the Network & Transport layer protocol. Also feedback from lower layers e.g. through Decentralized Congestion Control (DCC), or MCO, is used to adjust to the number of available resources. In case safety-critical data would have to be omitted to comply with the restrictions regarding message size or available resources, the data shall be segmented by `Mim` containers and two or more MIMs shall be generated to convey the data. This can for example be done by segmenting the data by driving regions or by car manufacturers.

NOTE: As an example, the `controlInterface` of a Mim container (clause 7.5) may be omitted, or the number of `wayPoin`t elements in the `pathControl` (clause 7.6.3.3) may be adapted to satisfy the message size and duty cycle limits.

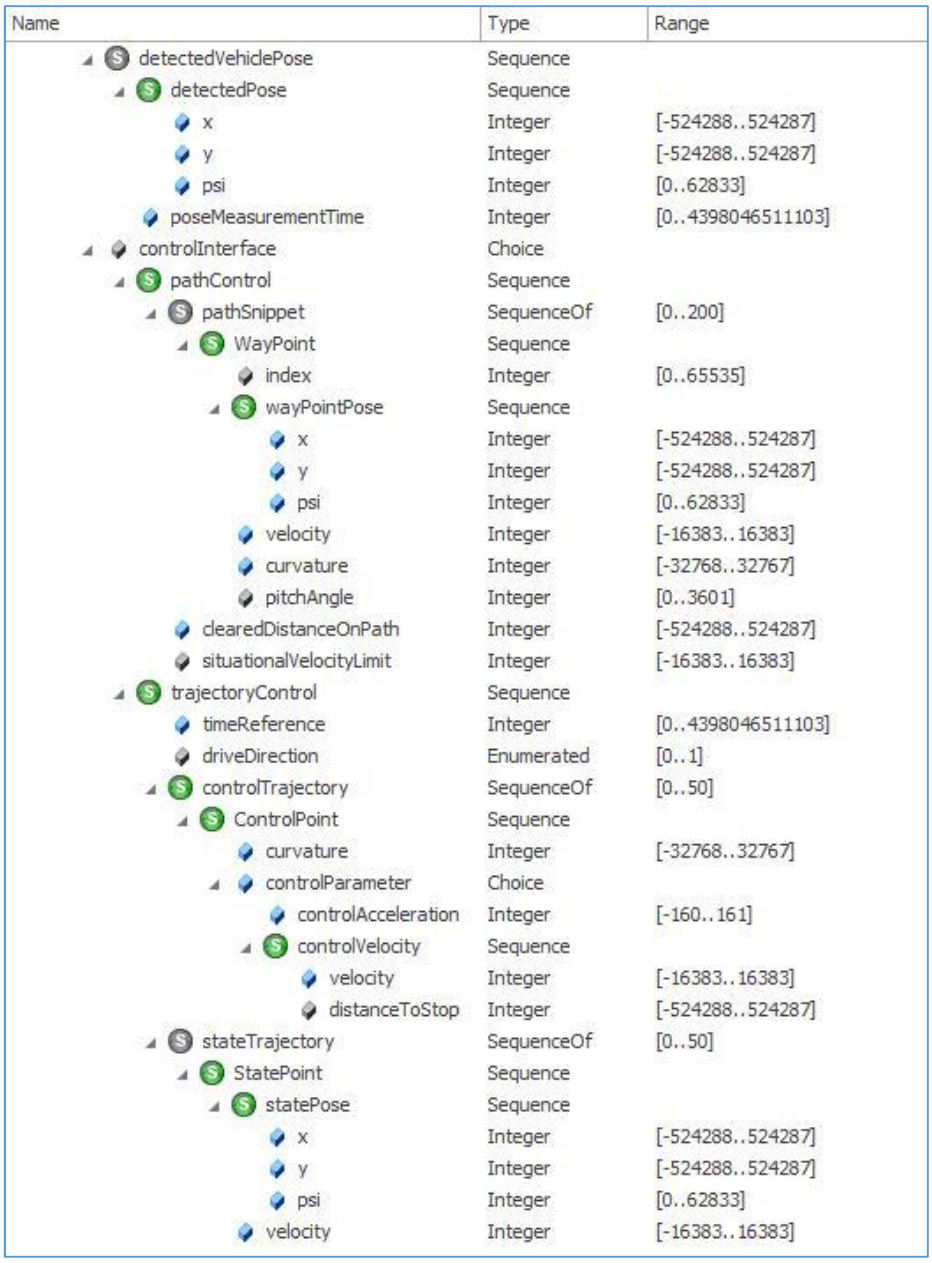
7 MIM containers and data elements

7.1 MIM structure overview

The MIM is the message which carries the Remote Operation RO commands and addresses the SVs. The hierarchical representation of the `Mim` containers and its data elements is portrayed in Figure 7 and Figure 8 and together build a summary of the complete MIM message. The depicted format is generated from eXtensible Markup Language (XML) description with courtesy of Vector Informatik GmbH.

Name	Type	Range
header	Sequence	
protocolVersion	Integer	[0..255]
messageId	Integer	[0..255]
stationId	Integer	[0..4294967295]
e2eProtection	Sequence	
length	Integer	[0..65535]
rollingCounter	Integer	[0..65535]
dataID	Integer	[0..4294967295]
crc32	Integer	[0..4294967295]
mims	SequenceOf	[1..32]
Mim	Sequence	
mimDataControlField	Sequence	
checksum	Integer	[0..4294967295]
mimGenerationTime	Integer	[0..4398046511103]
rollingCounterFromMvm	SequenceOf	[0..10]
UInt16	Integer	[0..65535]
proprietaryExtensionField	Integer	[0..65535]
systemManagementData	Sequence	
sessionID	IA5String	[17..32]
missionID	IA5String	[17..32]
vehicleID	IA5String	[1..17]
facilityID	IA5String	[1..32]
vehicleIdentification	Choice	
blinking	Sequence	
vidRoPublicKey	Integer	[0..18446744073709551615]
codeLength	Integer	[0..255]
blinkingCommand	Enumerated	[0..3]
drivingPermission	Sequence	
expirationTime	Integer	[0..4398046511103]
velocityMax	Integer	[-16383..16383]
curvatureMin	Integer	[-32768..32767]
curvatureMax	Integer	[-32768..32767]
checksum	Integer	[0..4294967295]
safetyTimeSyncRequest	Sequence	
challenge	Integer	[0..65535]
checksum	Integer	[0..4294967295]
driveCommand	Sequence	
driveCommandAction	Enumerated	[0..4]
terminateReason	Enumerated	[0..5]
gearRequest	Enumerated	[0..4]
directionIndicatorRequest	Enumerated	[0..4]
parkingBrakeRequest	Enumerated	[0..1]
motorSystemRequest	Enumerated	[0..2]
emergencyStopRequest	Enumerated	[0..4]
interlockRequest	Enumerated	[0..2]
hornRequest	Enumerated	[0..3]

Figure 7: MIM message structure (Part 1)



Name	Type	Range
detectedVehiclePose	Sequence	
detectedPose	Sequence	
x	Integer	[-524288..524287]
y	Integer	[-524288..524287]
psi	Integer	[0..62833]
poseMeasurementTime	Integer	[0..4398046511103]
controlInterface	Choice	
pathControl	Sequence	
pathSnippet	SequenceOf	[0..200]
WayPoint	Sequence	
index	Integer	[0..65535]
wayPointPose	Sequence	
x	Integer	[-524288..524287]
y	Integer	[-524288..524287]
psi	Integer	[0..62833]
velocity	Integer	[-16383..16383]
curvature	Integer	[-32768..32767]
pitchAngle	Integer	[0..3601]
clearedDistanceOnPath	Integer	[-524288..524287]
situationalVelocityLimit	Integer	[-16383..16383]
trajectoryControl	Sequence	
timeReference	Integer	[0..4398046511103]
driveDirection	Enumerated	[0..1]
controlTrajectory	SequenceOf	[0..50]
ControlPoint	Sequence	
curvature	Integer	[-32768..32767]
controlParameter	Choice	
controlAcceleration	Integer	[-160..161]
controlVelocity	Sequence	
velocity	Integer	[-16383..16383]
distanceToStop	Integer	[-524288..524287]
stateTrajectory	SequenceOf	[0..50]
StatePoint	Sequence	
statePose	Sequence	
x	Integer	[-524288..524287]
y	Integer	[-524288..524287]
psi	Integer	[0..62833]
velocity	Integer	[-16383..16383]

Figure 8: MIM message structure (Part 2)

Figure 9 shows the nested representation of the containers available in the MIM message. The colour scheme shows mandatory containers in green and optional ones in grey colour. The `ItsPduHeader` described in clause C.1 is shown in black and selectable containers (CHOICE) are in yellow. When building a message, selectable containers become mandatory during compilation once the selection is defined.

The base structure of the MIM message consists in the following containers:

1) The header `ItsPduHeader` container. (See clause C.1)

2) The end-to-end protection `e2eProtection` container. (See clause 7.2)

3) At least one vehicle `Mim` container. (See clause 7.4)

The ASN.1 representation of the Facilities layer message MIM shall follow the specifications as defined in Recommendation ITU-T X.680 [1].

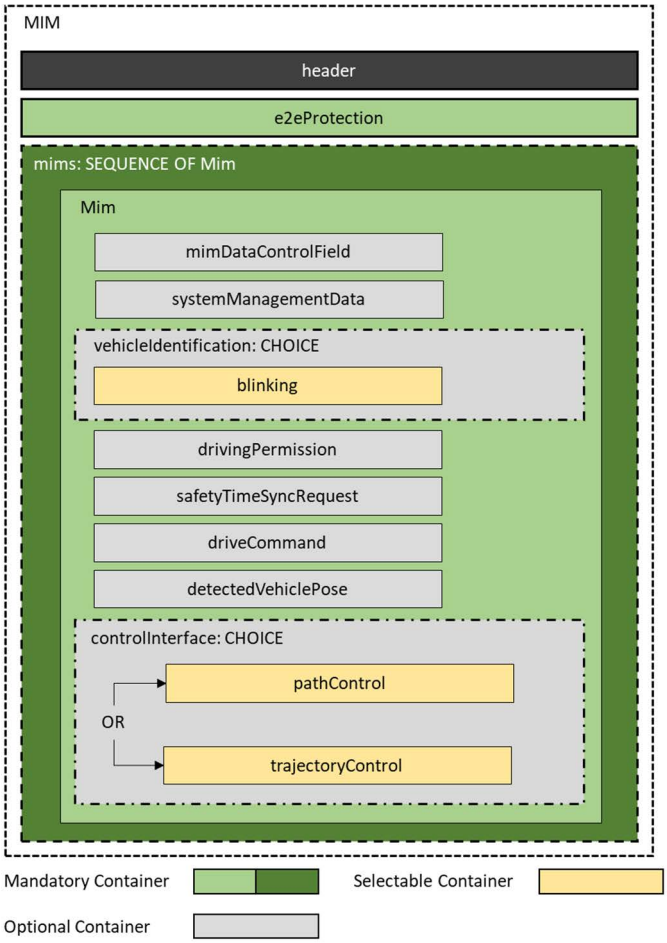


Figure 9: Containers structure within the MIM message

7.2 e2eProtection (container: AvmE2Eprotection)

7.2.1 End to end protection of the `Mim` container content

The structure of the `e2eProtection` container is portrayed in Figure 10 and shows the data elements to support the implementation of E2E protection mechanisms for the `Mim` container. These data elements are described in clauses 7.2.2, 7.2.3, 7.2.4 and 7.2.5 according to the AUTOSAR E2E Protocol Specification [i.6].

The data elements *length*, *counter*, *CRC32* and *DataID* defined in the `e2eProtection` container are based on the Autosar E2E profile 4 specification. As described in ISO 23374-1 [i.6], communication errors which are detected at lower layer will lead to discarding the related message. Thus, these messages might not reach the application at all. However, the end-to-end protection data elements allow developers to implement data protection mechanisms at the facilities layer between end points in an AVM architecture as shown in Figure D.1 in Annex D. Detailed instructions and an example of the use of the `e2eProtection` container to calculate data elements *length* and *CRC32* are provided in clause D.3.

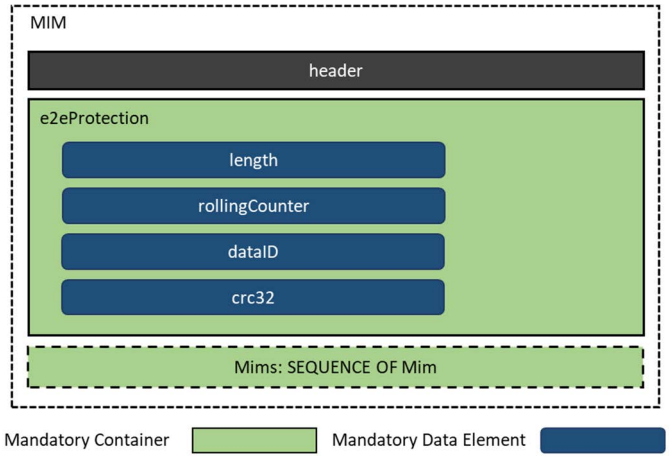


Figure 10: Structure of the *e2eProtection* container within the MIM message

7.2.2 *length* data element

The data element *length* represents the overall size in octets over that the protection mechanism is applied. The `ItsPduHeader` length (6 bytes) are subtracted from the complete length of the MIM.

7.2.3 *rollingCounter* data element

The data element *rollingCounter* is an identifier of a MIM message. For the first generation request the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent generation request of a MIM message. In case of overflow due to the size limit of the data element (0xFFFF) the counter shall restart with 0 for the next generation request.

On the SV side, the comparison of the counter of the received MIM against the counter of the previous MIM allows detection of the following conditions:

- Repetition: if the counters are the same.
- Communication error: if the counter increment is above a certain limit implying too many data lost.
- Normal condition: if the increment is by one or more than one but within the tolerated limits.

7.2.4 *dataID* data element

The data element *dataID* supports the verification of each transmitted MIM message at the SV side. *dataID* is to the best effort unique within the network of communicating systems as defined in [i.2].

In the VO subsystem, the comparison of the *dataID* of the received MIM against the expected *dataID* allows detection of the following conditions:

- Insertion of information: to verify that no additional information is added.
- Masquerading: to verify that no information is blanked out.
- Incorrect addressing: to verify whether an unknown *dataID* is transmitted.

7.2.5 *crc32* data element

As defined in [i.2], the Cyclic Redundancy Check is used in the VO subsystem to determine if bits flipped during the transmission of the MIM and thus the MIM has to be regarded as corrupted.

The calculation of CRC data element applies to the boundaries of the MIM bit stream while the starting 6 bytes from the `ItsPduHeader` are ignored. If necessary, the bit stream shall be enhanced with padding bits to result in an octet stream.

The CRC used in AUTOSAR E2E Profile 4 is a 32 bits CRC as described in [i.3].

7.3 Mims: SEQUENCE OF Mim

The *mims* sequence of the `Mim` containers after the `e2eProtection` provides a sequence of optional containers mainly to transmit commands and instructions from an RO subsystem to the VO subsystem. A MIM message shall include at least one `Mim`, thus the minimum number of vehicles to be addressed is 1 as shown in Figure 11.

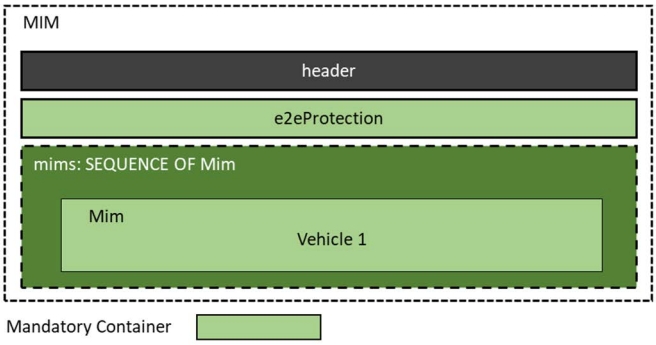


Figure 11: Minimal structure of the MIM message

A MIM may address one SV in unicast mode and may address several vehicles in broadcast mode. In broadcast mode all data elements described in clauses 7.2 and C.1 are relevant for all addressed SVs. A single MIM may contain n `Mim` containers as shown in Figure 12. The maximum number of addressable vehicles (n) depends on the overall size limits of the complete MIM message MTU_MIM (e.g. 1 600 bytes) and on the usage of optional containers and their data elements.

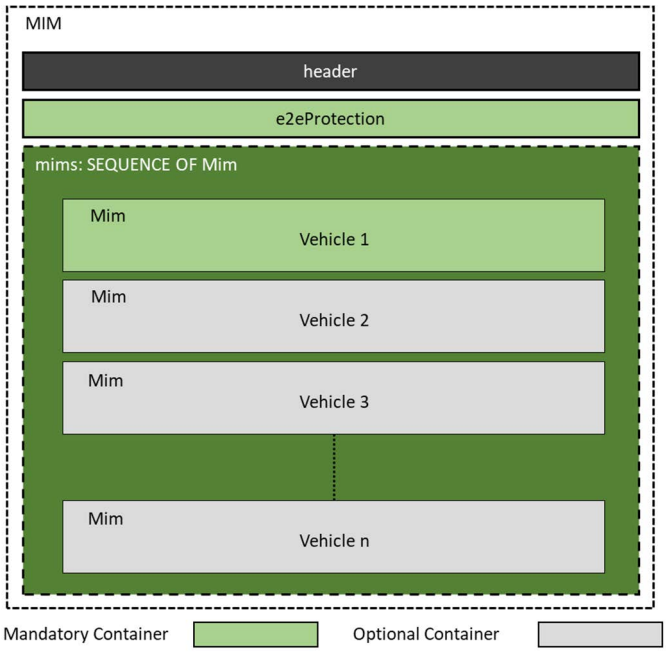


Figure 12: Concept of multiple `Mim` containers addressing multiple SVs

7.4 General Structure of mim (container: Mim)

7.4.1 Overview

The structure of a single `Mim` container is illustrated in Figure 9. A single `Mim` is assembled as a set of optional containers, each container supporting some functionality. Available optional containers are:

1) `mimDataControlField`
2) `systemManagementData`
3) `vehicleIdentification`
4) `drivingPermission`
5) `syfetyTimeSyncRequest`
6) `driveCommand`
7) `detectedVehiclePose`
8) `controlInterface`

7.4.2 mimDataControlField (container)

7.4.2.1 Introduction

The `mimDataControlField` container is an optional container. It consists of data elements mainly for the purpose of controlling the bidirectional communication between the VO subsystem and the RO subsystem.

Data elements in `mimDataControlField` as shown in Figure 13 are:

1) *checksum*
2) *mimGenerationTime*
3) *rollingCounterFromMvm as a sequence of RollingCounter*
4) *proprietaryExtensionField*

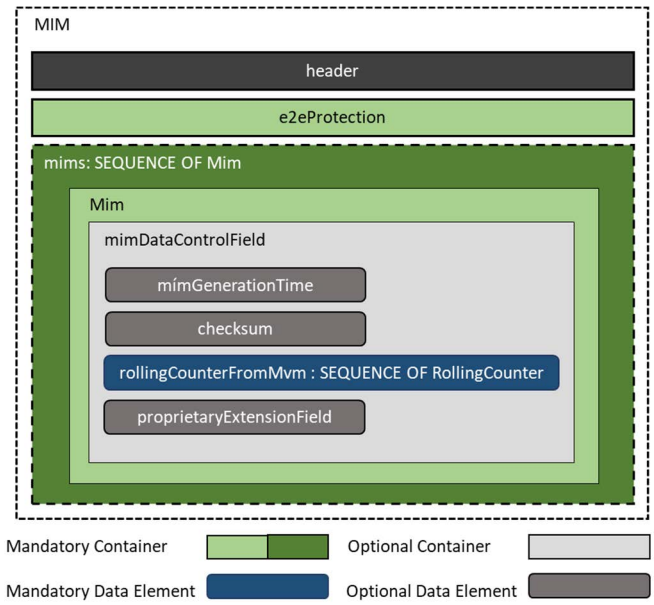


Figure 13: Structure of the `mimDataControlField` container within a MIM

7.4.2.2 *checksum* data element (optional)

The *checksum* data element is optional and serves as supplementary mean to support end-to-end protection. The checksum is calculated over a single vehicle container. Calculation rules and references like in clause 7.2 are not specified but are to be handled individually by the implementer.

7.4.2.3 *mimGenerationTime* data element (optional)

The *mimGenerationTime* data element is optional. It is used for checking the data freshness on the receiving side from message generation time included as part of the message transmission from the transmitter side.

7.4.2.4 *rollingCounterFromMvm*: SEQUENCE OF *Counter*

The data element *rollingCounterFromMvm* serves as a mirror of the rolling counter that was received with the latest, up to 10 MVM messages of the corresponding vehicle. It enables the vehicle system to recognize whether the RO has received fresh data from the vehicle or if there are lost messages in the radio transmission.

7.4.2.5 *proprietaryExtensionField* (optional element)

This data element defines 2 bytes that are used to carry specific information or request from an RO subystem to a specific OEM vehicle, e.g. an information that the vehicle now enters a car wash station or a request to open the charging flap at an electric charging station.

The bytes are unformatted, i.e. the format is only defined between the carmaker and the infrastructure provider in order to provide additional specific information in the driving protocol.

7.4.3 systemManagementData (optional)

7.4.3.1 Introduction

The `systemManagementData` container is portrayed in Figure 14. It consists of a series of optional data elements supporting to exchange the identification labels between the VO subsystem and the RO subsystem.

Available data elements are:

1) *sessionID*
2) *missionID*
3) *vehicleID*
4) *facilityID*

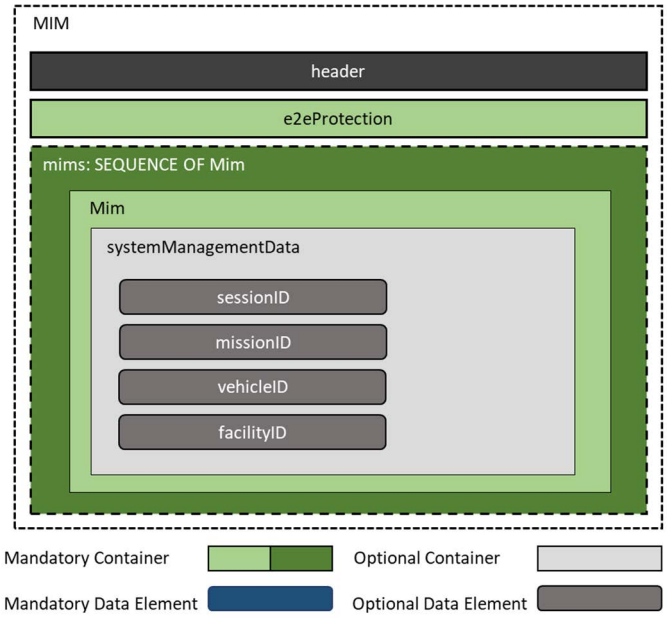


Figure 14: Structure of the `systemManagementData` container within a MIM

7.4.3.2 *sessionID* data element

The *sessionID* data element is a unique identifier to a sequence of interactions for a given SV between check-in and check-out. It identifies a set of multiple tasks or *missionID*s, i.e. a sequence of driving manoeuvres between a check-in and a check-out. It is negotiated and agreed upfront by both participants. Additional information e.g. timing validity, consensus of vehicle condition and body size, a suitable parking location is the part of upfront communication.

7.4.3.3 *missionID* data element

The *missionID* data element is a unique identifier that is known by the RO subsystem and the vehicle side at the same time. It identifies and describes a task that is negotiated and agreed upfront on both sides, e.g. driving from a parking location (origin) to the destination for a particular purpose like electric charging, washing or parking.

As defined in ISO 23374-1 [i.6], there may be more than one mission during one valid session period, but multiple missions are not carried out simultaneously for one SV.

7.4.3.4 *vehicleID* data element

The *vehicleID* data element is an optional identifier. As several sessions can in principle be related to the same vehicle over time, this data element makes sure that the RO subsystem can uniquely identify vehicles. This also allows the RO to deal with vehicle parameters, as the Vehicle Backend (VB) can communicate a list of *vehicleID* values and their associated parameters before operation, and this will be independent from *sessionID* values. The vehicle needs to know its own *vehicleID*. This can serve as a safety check for the RO regarding vehicle parameters, as it receives them from the VB, and it can then verify that it is communicating to the right vehicle through MVM (as the vehicle sends its *vehicleID* there). Another option for this check would be for the vehicle to verify that the *vehicleID* it receives via MIM corresponds to its own. Having *vehicleID* in both messages leaves these two options open. The format of *vehicleID* is similar to VIN. The OEM could in principle decide to use real VIN or fake ones, as long as the identifier is uniquely defined.

7.4.3.5 *facilityID* data element

The data element *facilityID* provides the option for the identification of the infrastructure facility, e.g. a parking lot or a logistics depot, depending on the use case. It serves for documentation purposes and should be identical in the corresponding MIMs and MVMs.

7.4.4 vehicleIdentification: CHOICE

7.4.4.1 Introduction

The `vehicleIdentification` allows the selection of the mechanism for physical identification of the SV by the sensors of the AVM system. The blinking method using the SV blinkers is available but alternative identification methods may be defined in the future as depicted in Figure 15.

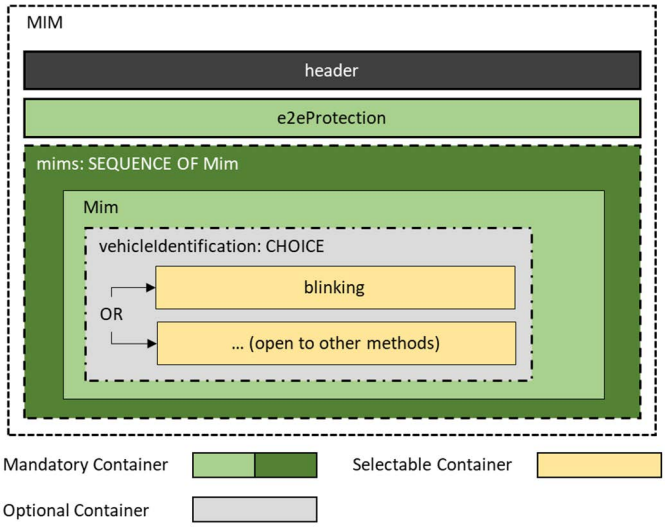


Figure 15: Structure of the `vehicleIdentification` container within the MIM message

7.4.4.2 blinking (container: Blinking)

7.4.4.2.1 Introduction

The `blinking` container shown in Figure 16 carries signals that transmit a code to the vehicle to generate a blinking pattern for the direction indicator lights. This light pattern transmitted by the vehicle is used by the infrastructure sensors to identify the according vehicle at its expected starting or drive-off location.

5GAA TR-22002 [i.8] and in more detail in the VDA Position Document [i.7] describe for this purpose a seed process. In order to support broadcast communication, it differs from the key exchange process in this clause.

The seed is negotiated using a Diffie-Hellman key exchange. The public key exchange process allows the two participants to agree on a secret seed. It is combined with the optional container `VidResponse` in MVM message, see clause 8.3.5.

In return the key code is transmitted by the vehicle in form of a blinking pattern of the indicator lights. The RO subsystem receives this pattern by camera sensing. If the key code is verified RO continues to control the vehicle's automated drive procedure and VMC.

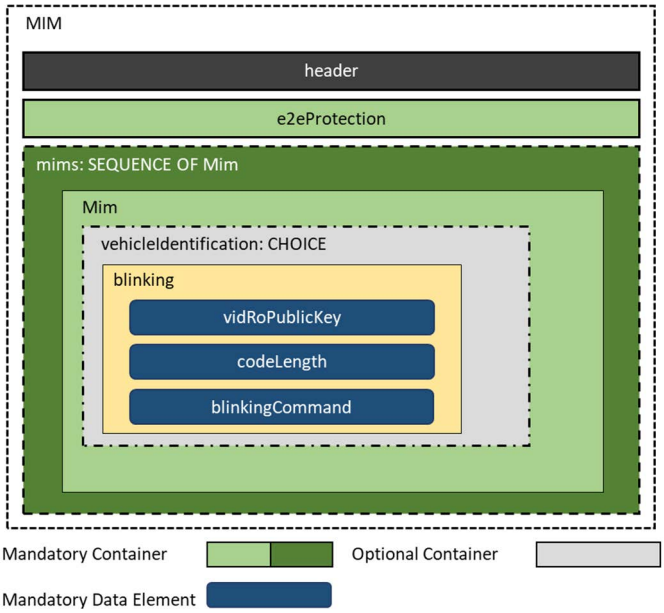


Figure 16: Structure of the `blinking` container within the MIM message

7.4.4.2.2 *vidRoPublicKey* data element

Using the *vidRoPublicKey*, a 64-bit public key is transmitted from the RO subsystem to the VO subsystem to derive the SV identification secret.

7.4.4.2.3 *codeLength* data element

The *codelength* data element sends a value that indicates how many bits (8 to 20) from the *vidRoPublicKey* shall be used for generating the blinking pattern.

7.4.4.2.4 *blinkingCommand* data element

This data element mirrors the current identification request status from the perspective of the RO system. The *blinkingCommand* values are shown in Table 2.

Table 2: *blinkingCommand* values

State	Description
generateNewCode	A new safe vehicle identification cycle was started with no intent to flash.
generateNewCodeAndPrepareForFlashing	A new safe vehicle identification cycle was started. Flashing the indicator light is required in this cycle.
flashing	The infrastructure is prepared and waiting for the Subject Vehicle to flash the code.
successful	The Subject Vehicle was recognized correctly and the identification is completed.

7.4.5 drivingPermission (container: DrivingPermission)

7.4.5.1 Introduction

The `drivingPermission` concept offers an expiration time for `drivingPermission` and the boundary values of longitudinal and lateral movement of the SV. The time synchronization between RO subsystem and VO subsystem is a binding prerequisite to apply the concept of expiration time. A detailed description of the `drivingPermission` concept is referenced in 5GAA TR T-220002 [i.8].

The `drivingPermission` container is flagged as optional, there is no explicit need to send the container to the addressed vehicle with every MIM message. The concept is also optionally used as alternative to the data element *emergencyStopRequest* in clause 7.4.7.8

Figure 17 shows the data elements within the `drivingPermission` container.

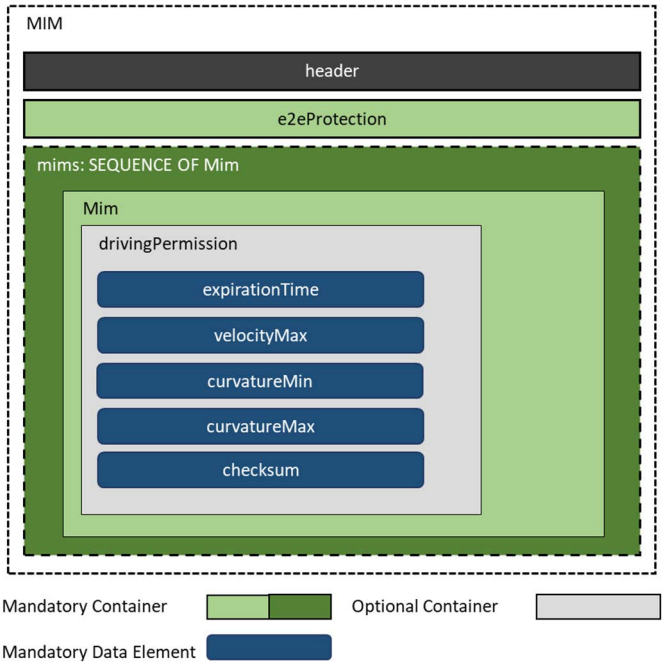


Figure 17: Structure of the `drivingPermission` container within the MIM message

7.4.5.2 expirationTime data element

The data element *expirationTime* represents an absolute time stamp, aligned with the time synchronization process between RO subsystem and the VO subsystem. It indicates that the vehicle should start braking when the expiration time is reached.

7.4.5.3 *velocityMax* data element

This DE shall indicate that the vehicle should not exceed the maximal velocitiy value *velocityMax.*

7.4.5.4 curvatureMin and curvatureMax data elements

These data elements shall indicate that the vehicle should not exceed the maximal and minimal curvature values velocitiy value *curvatureMax* and *curvatureMin.*

7.4.5.5 *checksum* data element

The container `drivingPermission` shall be protected by a dedicated checksum to ensure end-to-end protection against accidental changes between the safety components in the SV and in the AVM system. An example on how to calculate data element checksum is provided in the informative clause D.4.3.

7.4.6 safetyTimeSyncRequest (container: SafetyTimeSyncRequest)

7.4.6.1 Introduction

To calculate the ITS timestamp given in `drivingPermission:`*expirationTime*, the RO subsystem needs to be able to continuously determine the current time in which the VO subsystem of an SV operates.

This means that the RO subsystem determines the offset between the RO clock and the VO clock. Based on this offset it is able to determine the time of the VO subsystem in an SV.

The structure of the `safetyTimeSyncRequest` container and its data elements is shown in Figure 18.

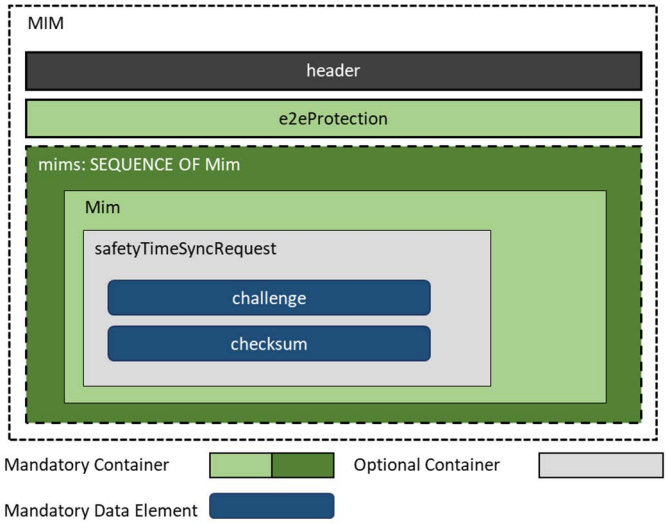


Figure 18: Structure of the *safetyTimeSyncRequest* container within the MIM message

7.4.6.2 *challenge* data element

The DE *challenge* is chosen by the RO subsystem in accordance with the safety requirements. The RO subsystem shall use this challenge to relate the response from the VO subsystem to the *safetyTimeSyncRequest*.

7.4.6.3 *checksum* data element

The *`safetyTimeSyncRequest`* needs to be protected by a dedicated safety checksum transmitted using the data element *checksum*. An example how to calculate data element checksum is provided in clause D.4.2.

7.4.7 driveCommand (container: DriveCommand)

7.4.7.1 Introduction

The drive command is core of the MIM messaging. It addresses the specific vehicle and determines the actual state request to the vehicle, especially the active drive request for the overall VMC operation.

The structure of the `driveCommand` container within the MIM message is shown in Figure 19.

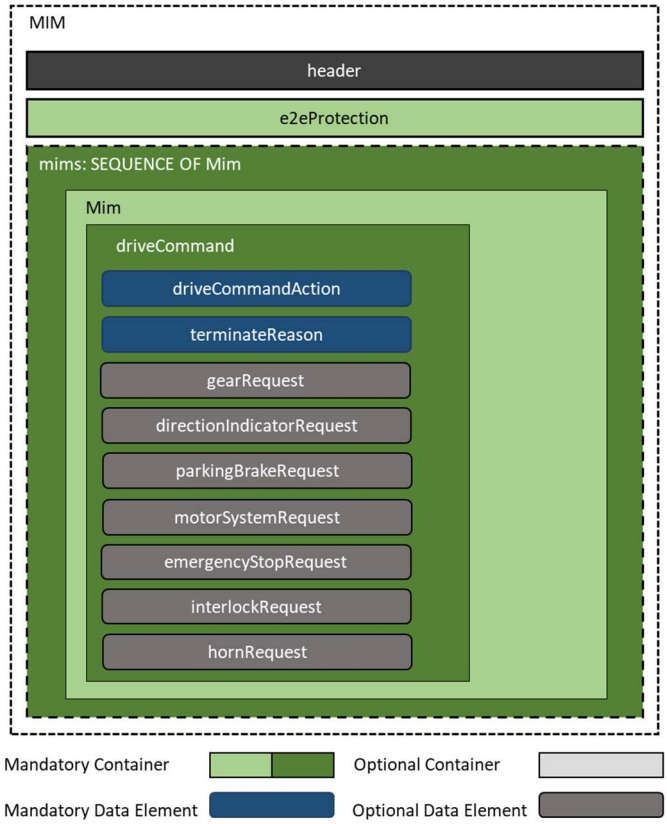


Figure 19: Structure of the `driveCommand` container within the MIM message

7.4.7.2 driveCommandAction data element

The data element *driveCommandAction* offers the states shown in Table 3.

Table 3: *driveCommandActions* states

State	Description
sleep	The vehicle is commanded to go into sleep mode for low power consumption.
initialize	The vehicle remains in standstill and shall initialize and prepare for a driving job, including a sequence of power-on the engine system and the lights. The vehicle identification process can be executed in this stage.
wait	The vehicle shall wait in marshalling capable state.
drive	The vehicle is commanded to actively drive and follow the according control commands. Standstill and pause situations are potentially included in this stage.
terminate	The vehicle shall disable the according control interface. After brining the vehicle to a safe standstill state it shuts down as soon as possible.

7.4.7.3 terminateReason data element

The data element *terminateReason* offers the states presented in Table 4.

Table 4: *terminateReason* states

State	Description
proceed	Everything is okay. Proceed, do not terminate.
destinationReached	Vehicle has reached its destination.
infrastructureError	Error in infrastructure.
vehicleError	Vehicle has sent an error code.
backend	Error in backend.
vehicleIdentificationError	Either a wrong or no code was received during the vehicle identification process.

7.4.7.4 *gearRequest* data element (optional)

Data element *gearRequest* commands the gear selection in the SV and it is an optional container. It offers the following requests shown in Table 5.

Table 5: g*earRequest* commands

State	Description
park	P: Vehicle shall engage P.
backwards	R: Vehicle shall drive backwards.
neutral	N: Vehicle shall engage neutral gear.
forwards	D: Vehicle shall drive forwards.
unknown	Not used for commanding a vehicle gear.

7.4.7.5 directionIndicatorRequest data element (optional)

Data element *directionIndicatorRequest* offers the commands shown in Table 6.

Table 6: *diretionIndicatorRequest* commands

State	Description
off	Do not flashlights.
right	Flash right.
left	Flash left.
both	Flash left and right.
unknown	Direction indicator request is undefined.

7.4.7.6 parkingBrakeRequest data element (optional)

Data element *parkingBrakeRequest* offers the commands shown in Table 7.

Table 7:*parkingBrakeRequest* commands

State	Description
disengage	The vehicle shall disengage the electric parking brake.
engage	The vehicle shall activate the electric parking brake.

7.4.7.7 motorSystemRequest data element (optional)

Data element *motorSystemRequest* offers the commands shown in Table 8.

Table 8: *motorRequest* commands

State	Description
off	The propulsion motor: off.
on	The propulsion motor: on.
umknown	The propulsion motor state is unknown (not used as a command).

7.4.7.8 *emergencyStopRequest* data element (optional)

This data element enables as part of an optional safety concept to apply a vehicle specific emergency stop manoeuvre at any time, even outside of a regular message periodicity of MIM. It is initiated by the infrastructure and executed by the according vehicle, also to support the pre-charge of the hydraulic brake system. This data element offers the commands described in Table 9.

Table 9: *emergencyStopRequest* commands

State	Description
inactive	The vehicle does not use emergencyStop at this stage.
precharge	The vehicle applies a hydraulic brake pre-charge (to overcome the air gap of brake pads and calliper) without decelerating the vehicle. At this stage the vehicle waits for an active command.
active	The vehicle bake system initiates an emergency stop using the maximum possible deceleration.
tempError	Command for maturing failures in RO subsystem. The vehicle shall decelerate and hold, waiting for failure to clear or mature.
suspend	Command for critical failures (not recoverable) in RO subsystem. The vehicle shall decelerate and secure.

7.4.7.9 *interlockRequest* data element (optional)

This data element enables an optional safety concept to apply a vehicle interlock. This data element offers the commands described in Table 10.

Table 10: *interlockRequest* commands

State	Description
none	The vehicle does not use interlock at this stage.
zonalInterlock	The vehicle applies a zonal interlock function.
globalStop	The vehicle as to stop for intervention of an (external human) operator.

7.4.7.10 *hornRequest* data element (optional)

This data element enables an optional safety concept. It commands the vehicle to sound the horn in different formats in order to warn pedestrians or animals. This data element offers the states described in Table 11.

Table 11: *hornRequest* states

State	Description
none	The vehicle does not sound a horn at this stage.
singleHorn	The vehicle applies a single horn for alerting surroundings when marshalling of vehicle is starting.
doubleHorn	For notifying pedestrians who pose obstacles to the vehicle marshalling task.
holdHorn	In event of ground staff intervention needed for vehicle.

7.4.8 detectedVehiclePose (optional container: DetectedVehiclePose)

7.4.8.1 Introduction

The AVM system permanently senses the vehicle's position in a relative two-dimensional coordinate system (X and Y). The pointed vehicle position reference is the middle of the rear axle. The detected vehicle pose is reported in this data element and complemented with the vehicle's heading *psi* and a timestamp *poseMeasurementTime*. This container is optional, and it is usable for the path snippet control mechanism.

The structure of the `detectedVehiclePose` container within the MIM message is shown in Figure 20.

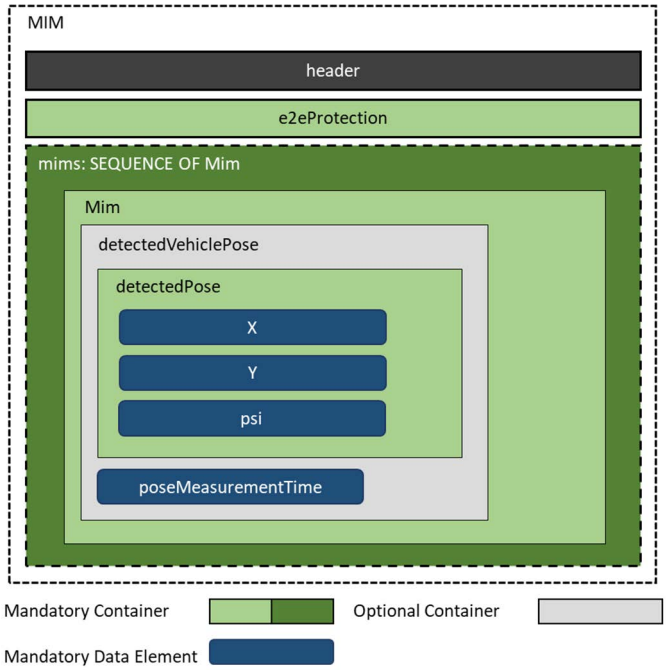


Figure 20: Structure of the `detectedVehiclePose` container within the MIM message

7.4.8.2 detectedPose (container: Pose)

This field implements the sequence of data elements `pose` described in clause 7.7.7.2.

7.4.8.3 poseMeasurementTime data element

The DE signals an absolute time stamp within the time synchronization of RO and VO. It represents when the RO measurement of the position was taken. It is aligned with the time synchronization process.

7.5 controlInterface: CHOICE

The protocol is offering two different control methods with variants (clause 4.3.4) to drive the vehicles remotely by commands of the infrastructure system. The control method is addressed as a CHOICE as described in the ASN.1 definitions for a MIM message. Herewith additional control methods are applicable in later versions of the protocol.

The structure of the `controlInterface` within the MIM message is shown in Figure 21.

The SO shall use only one methodology for AVM operation of a particular SV. That is, the VMC operation interface between the RO and VO, shall be either Path Control or Trajectory Control. If the VMC operation interface implements Trajectory Control, it shall use only one of the two available motion parameters.

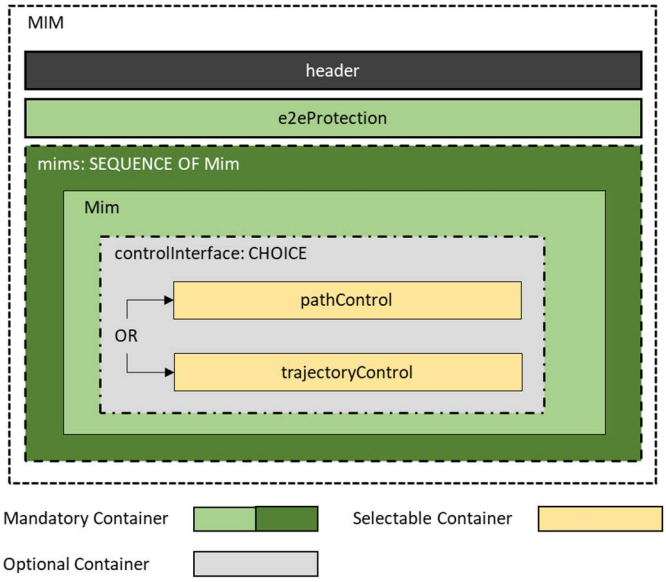


Figure 21: Structure of the `controlInterface` CHOICE within the MIM message

7.6 pathControl (container: PathControl)

7.6.1 Introduction

The `pathControl` container transfers the path snippets described in clause 7.6.2 from the RO into the VO.

The structure of the `pathControl` container within the MIM message is shown in Figure 22.

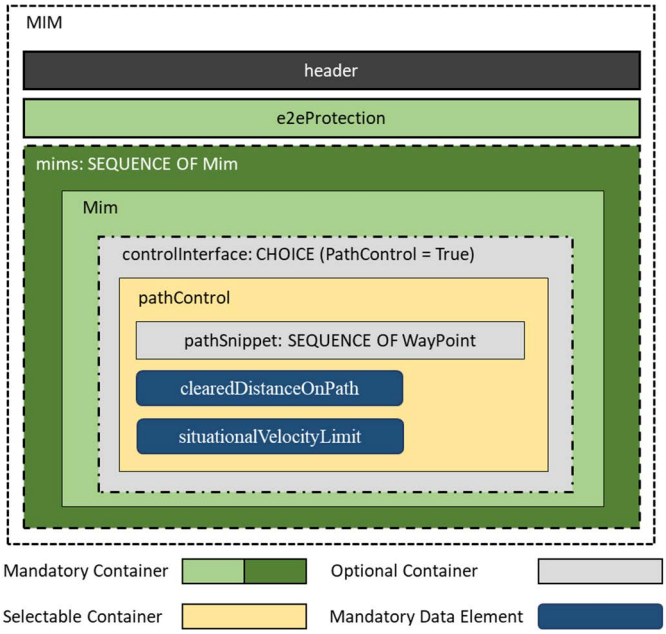


Figure 22: Structure of the `pathControl` container within the MIM message

7.6.2 PathSnippet: SEQUENCE OF WayPoint

The `pathSnippet` container transfers a sequence of `wayPoint` containers. It is possible to send a sequence of `wayPoint` elements. In accordance to clause 6.3 the number of `wayPoint` containers shall be chosen such that the generated MIM complies with *MTU_MIM*. This can be done, e.g. through segmentation of the generated PDU into smaller segments. The number of `wayPoint` containers shall further be chosen such that the generated MIM complies with the regulatory constraints of the selected communication channel, such as duty cycle or available data rate as provided by MCO.

The `pathSnippet` container is optional and a structure `pathControl`. `pathSnippet` containing no `wayPoints` has a different meaning than a structure `pathControl` with no given `pathSnippet`. When sending a `pathControl` sequence, the RO can omit repeating a previously sent `pathSnippet`. When doing so, it tells the SV that is should keep following the last known `pathSnippet`.

The structure of the `pathSnippet` container within the MIM message is shown in Figure 23.

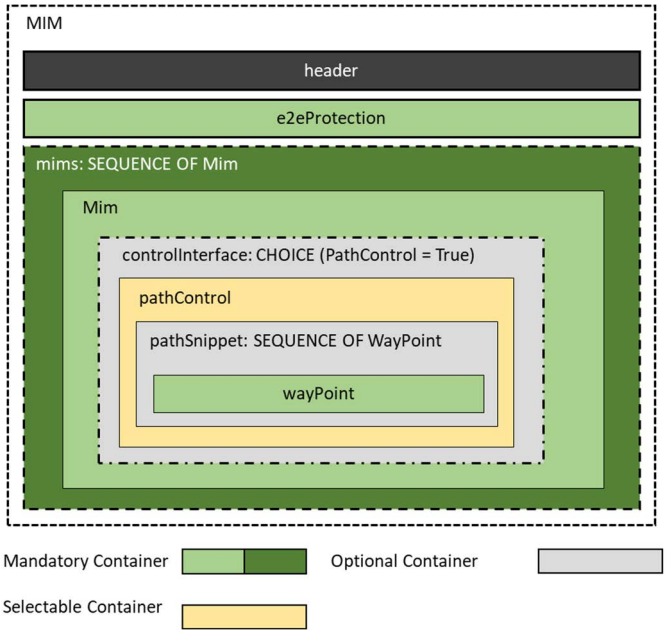


Figure 23: Structure of the `pathSnippet` container within the MIM message

7.6.3 wayPoint (container: WayPoint)

7.6.3.1 Introduction

The `wayPoint` container offers signal information like index, a two-dimensional position in a proprietary coordination system as well as the desired driving direction (heading). It also contains the requested speed and curvature commands.

The structure of the `wayPoint` container within the MIM message is shown in Figure 24.

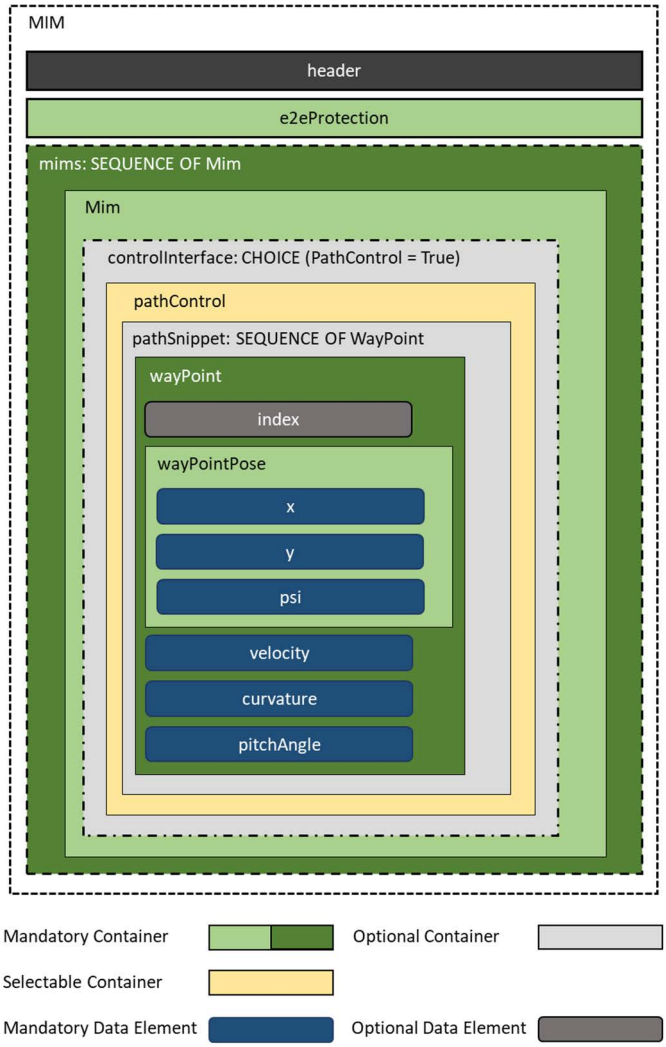


Figure 24: Structure of the *wayPoint* container within the MIM message

7.6.3.2 *index* data element (optional)

The index DE is a 2-byte value. It is used to build an array of consecutive way points, stored in the VO subsystem. Way points are updated with this index DE if a planned trajectory is to be changed.

7.6.3.3 wayPointPose (container: Pose)

This field implements the sequence of data elements described in clauses 7.7.7.2.2, 7.7.7.2.3 and 7.7.7.2.4.

7.6.3.4 *velocity* data element

This data element represents the target speed of the vehicle in that specific way point.

7.6.3.5 *curvature* data element

This data element represents the target curvature of the vehicle in that specific way point.

7.6.3.6 *pitchAngle* data element (optional)

This data element represents the road inclination on which of the vehicle drives in that specific way point. On inclined road segments this support the vehicle to better adopt the velocity control. The unit is cartesian angle value.

7.6.4 clearedDistanceOnPath data element

With this data element the RO can tell the vehicle to stop at the according distance on the known path snippet without sending a new path snippet. The unit of the DE is centimetre.

7.6.5 situationalVelocityLimit data element (optional)

With this data element the RO can tell the vehicle temporarily drive slower than indicated on the known path snippet without sending a new path snippet. The unit of the DE is m/s.

7.7 trajectoryControl (container: TrajectoryControl)

7.7.1 Introduction

A vehicle trajectory consists of a vector of control and state points transmitted from the RO to the SV using the `controlTrajectory` and `stateTrajectory` containers. This structure within a MIM is shown in Figure 25. The `trajectoryControl` container also contains a reference time stamp and an optional drive direction element. Details of this optional control methods are described in [i.7].

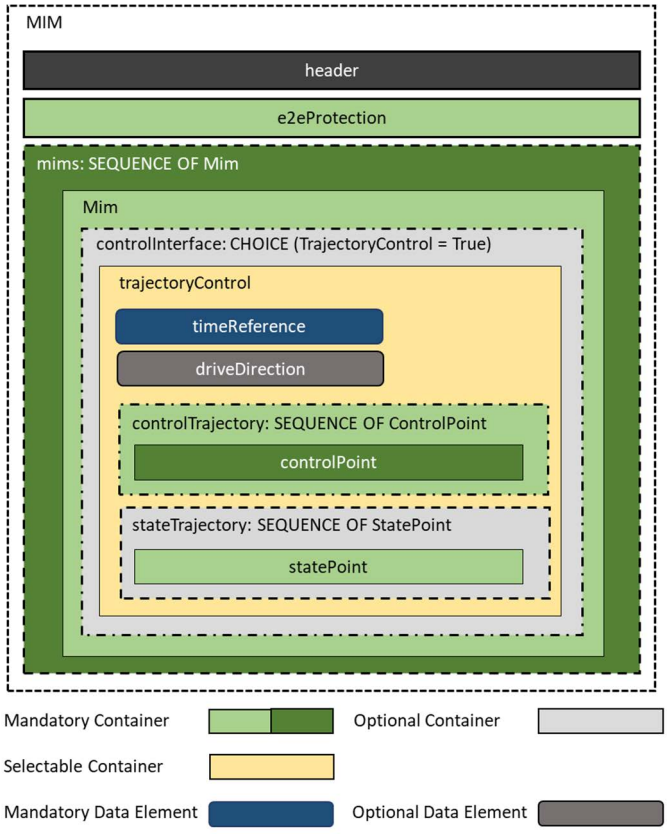


Figure 25: Structure of the `trajectoryControl` container within the MIM message

7.7.2 *timeReference* data element

The reference time indicates the absolute time given in Vehicle Functional Clock at which the first element of the `ControlTrajectory` vector and `StateTrajectory` vector is expected to be executed.

7.7.3 *driveDirection* data element

The data element *driveDirection* is optional. It describes the desired driving direction of the vehicle in alignment with the signed value of the vehicle acceleration requests as shown in Table 12.

Table 12: *driveDirection* states

State	Description
forwards	D: Vehicle shall drive forwards
backwards	R: Vehicle shall drive backwards

7.7.4 controlTrajectory: SEQUENCE OF ControlPoint

The `controlTrajectory` container consists of a sequence of `controlPoint` containers. In accordance to clause 6.3 the number of `controlPoint` containers shall be chosen such that the generated MIM complies with MTU_MIM. This can be done, e.g. through segmentation of the generated PDU into smaller segments. The number of `controlPoint` containers shall further be chosen such that the generated MIM complies with the regulatory constraints of the selected communication channel, such as duty cycle or available data rate as provided by MCO.

The structure of the `controlTrajectory` container within the MIM message is shown in Figure 26.

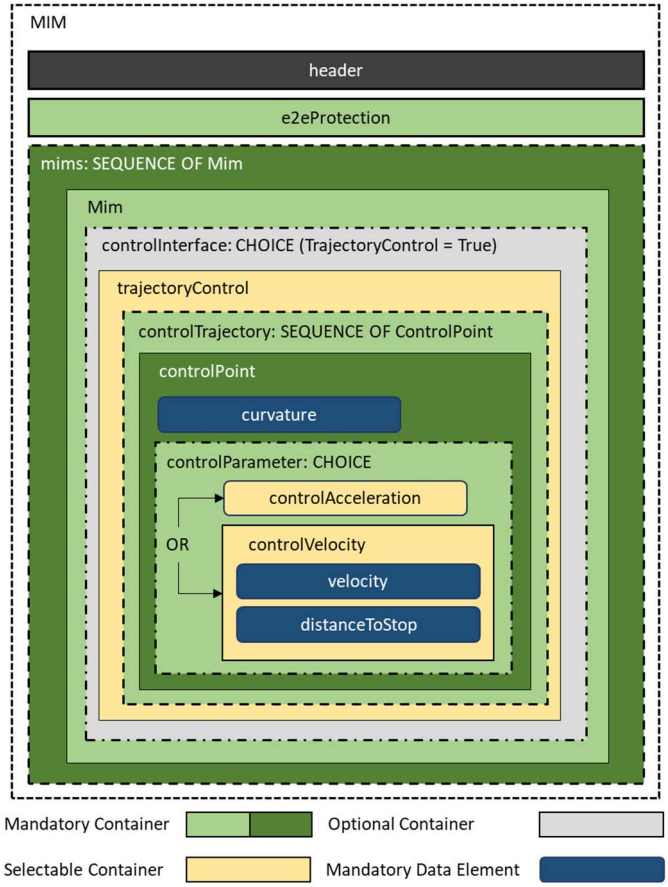


Figure 26: Structure of the `controlTrajectory` container within the MIM message

7.7.5 controlPoint (container: ControlPoint)

7.7.5.1 Introduction

A `controlPoint` container carries a set of control point elements: *curvature* as a data element and `controlParameter` as a CHOICE as depicted in Figure 26.

7.7.5.2 *curvature* data element

The data element represents the target curvature in the control point. The format of this data element is described by the element "HighResCurvature" in the AVM_commons.asn, see Annex A. It differs from the CDD element *curvatureValue* because a higher resolution is required, that comes with new steering angle sensor technologies.

7.7.5.3 controlParameter: CHOICE

The choice given to `controlParameter` allows selecting between longitudinal acceleration via *controlAcceleration* or longitudinal velocity via the container `controlVelocity`. The CHOICE structure is presented in Figure 26.

7.7.5.4 *controlAcceleration* data element

The data element represents the target acceleration in the control point as described with Figure 27. The format of this data element is adopted to the common data dictionary CDD data element "LongitudinalAccelerationValue". For details refer to ETSI TS 102 894-2 [i.11].

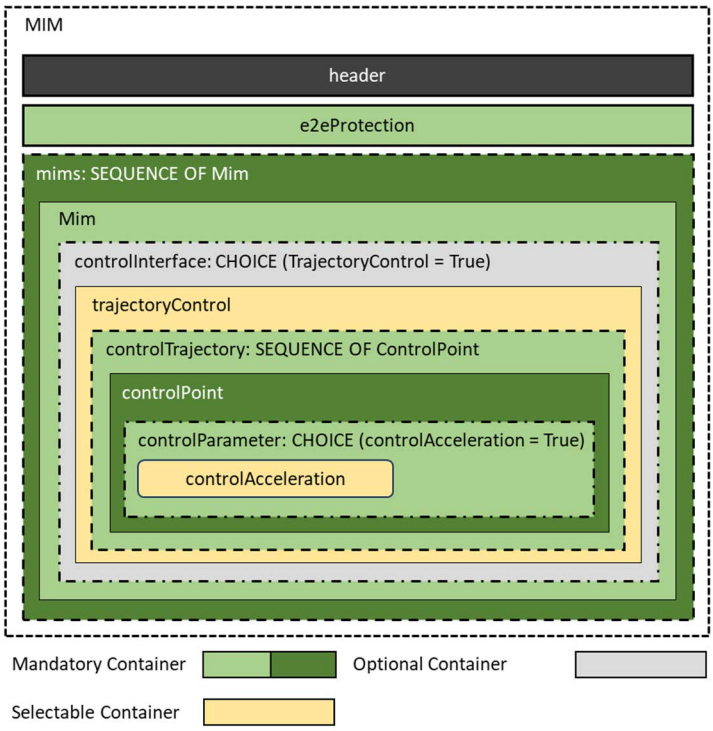


Figure 27: Structure of the `controlPoint` container for CHOICE *controlAcceleration* TRUE

7.7.5.5 controlVelocity (container:ControlVelocity)

7.7.5.5.1 Introduction

The container `controlVelocity` offers the data elements *velocity* and *distanceToStop* as shown in Figure 28.

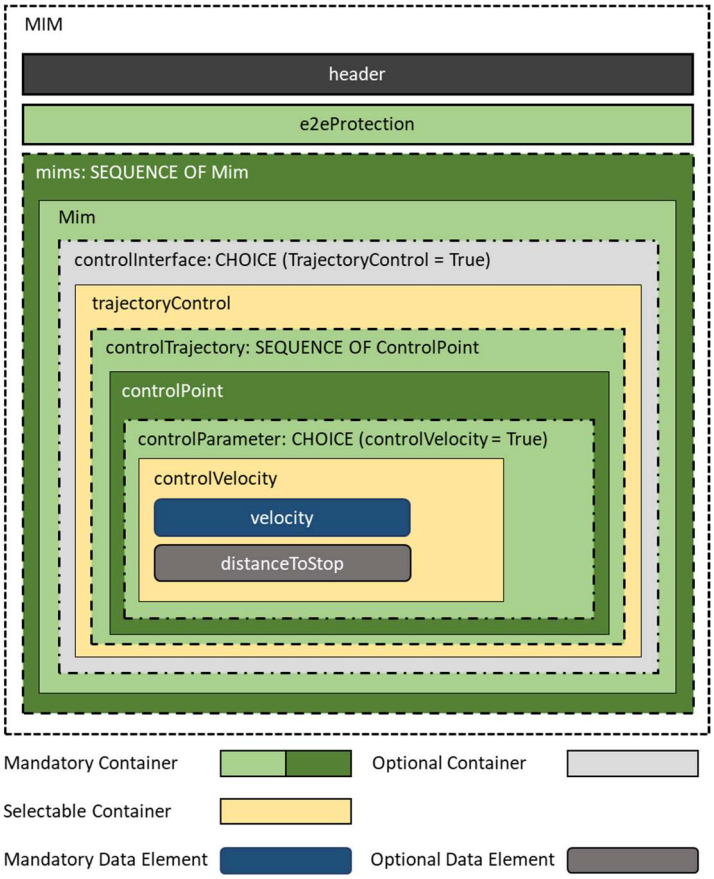


Figure 28: Structure of the `controlPoint` container for CHOICE *controlVelocity* TRUE

7.7.5.5.2 v*elocity* data element

The data element represents the target speed of the vehicle. It is a signed velocity value. The unit is cm/s. A positive value represents the forward drive of the vehicle, a negative value means reverse driving.

7.7.5.5.3 *distanceToStop* data element (optional)

The data element represents the maximum distance that the vehicle can drive before a standstill. Size for distance measurement is 20 bit, resolution is 1 cm. It is to use optionally in order to position the vehicle accordingly when it goes over ramps and wheel chocks.

7.7.6 stateTrajectory: SEQUENCE OF StatePoint

The elements in the `stateTrajectory` are considered as odometry target values. These consist of a sequence of `statePoint` containers. The number of `statePoint` containers shall be chosen such that the generated MIM complies with MTU_MIM. This can be done, e.g. through segmentation of the generated PDU into smaller segments. The number of `statePoint` containers shall further be chosen such that the generated MIM complies with the regulatory constraints of the selected communication channel, such as duty cycle or available data rate as provided by MCO. The structure of the `stateTrajectory` container within the MIM message is shown in Figure 29.

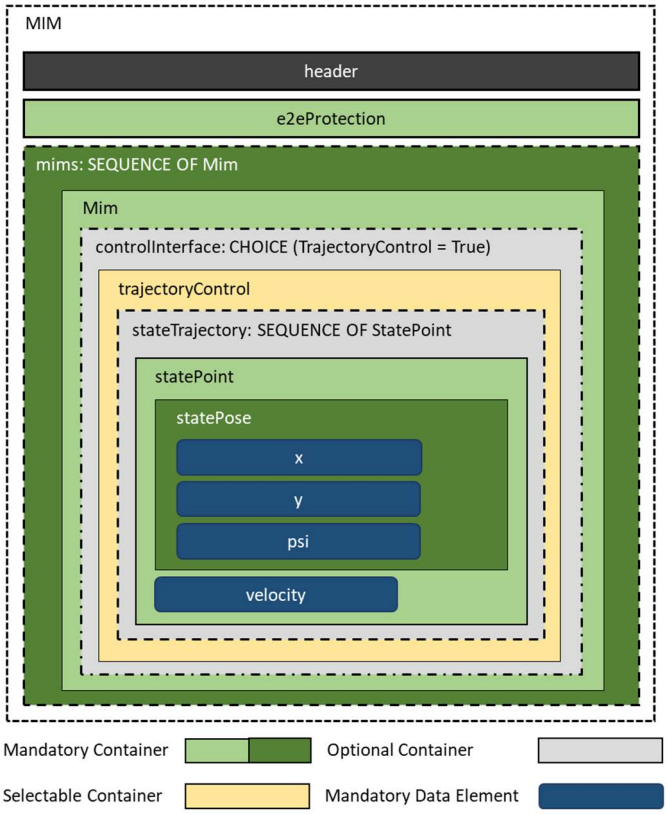


Figure 29: Structure of the `stateTrajectory` container within the MIM message

7.7.7 statePoint (container: StatePoint)

7.7.7.1 Introduction

A `statePoint` container carries a set of state point elements, offering the following elements: `statePose` as container and *velocity* as data element.

7.7.7.2 statePose (Container: Pose)

7.7.7.2.1 Introduction

The `statePose` container carries the target coordinates *x*, *y* and *psi* of the vehicle. The according reference point of a vehicle pose/position is an agreed point of the vehicle that is agreed between the RO and the vehicle VO e.g. the centre of the rear axle. The coordinate system is proprietary and established by the RO system design.

7.7.7.2.2 *x* data element

This data element is 20 bit for the x position of the way point. Resolution is 1 cm.

7.7.7.2.3 *y* data element

This data element is 20 bit for the y position of the way point. Resolution is 1 cm.

7.7.7.2.4 *psi* data element

This data element is 16 bits for the desired heading position of the vehicle in that specific way point. A heading (*psi*) value of 0 represents a drive in x-direction. The unit is in 0,0001 radian and counted counter-clockwise.

7.7.7.3 *velocity* data element

The data element represents the target speed of the vehicle. It is a signed velocity value. The unit is cm/s. A positive value represents the forward drive of the vehicle; a negative value means reverse driving.

8 MVM containers and data elements

8.1 MVM structure overview

The MVM is the message which elements describe the remote operation of the vehicle from the vehicle perspective. The hierarchical representation of the MVM containers and its data elements is portrayed in Figure 30 and Figure 31. The type and range information complies with ASN.1 Unaligned Packed Encoding Rules (UPER).

The containers and data elements represented in Figure 30 and Figure 31 together build a summary of the complete MVM message. The depicted format is generated from XML description with courtesy of Vector Informatik GmbH.

Name	Type	Range
header	Sequence	
protocolVersion	Integer	[0..255]
messageId	Integer	[0..255]
stationId	Integer	[0..4294967295]
e2eProtection	Sequence	
length	Integer	[0..65535]
rollingCounter	Integer	[0..65535]
dataID	Integer	[0..4294967295]
crc32	Integer	[0..4294967295]
mvm	Sequence	
mvmDataControlField	Sequence	
mvmGenerationTime	Integer	[0..4398046511103]
rollingCounterFromMim	SequenceOf	[0..10]
UInt16	Integer	[0..65535]
proprietaryExtensionField	Integer	[0..65535]
systemManagementData	Sequence	
sessionID	IA5String	[17..32]
missionID	IA5String	[17..32]
vehicleID	IA5String	[1..17]
facilityID	IA5String	[1..32]
vehicleState	Sequence	
vehicleStateGenerationTime	Integer	[0..4398046511103]
operationMode	Enumerated	[0..8]
gearState	Enumerated	[0..4]
directionIndicatorState	Enumerated	[0..4]
parkingBrakeState	Enumerated	[0..4]
motorSystemState	Enumerated	[0..2]
currentVelocity	Integer	[-16383..16383]
currentCurvature	Integer	[-32768..32767]
secureStandstill	Boolean	[false..true]
idxLastWayPoint	Integer	[0..65535]
localizedPose	Sequence	
x	Integer	[-524288..524287]
y	Integer	[-524288..524287]
psi	Integer	[0..62833]

Figure 30: MVM message structure (part 1: main containers)

Name	Type	Range
vidResponse	Sequence	
vidVehicleState	Enumerated	[0..5]
vidVehiclePublicKey	Integer	[0..18446744073709551615]
safetyTimeSyncResponse	Sequence	
challenge	Integer	[0..65535]
vehicleSafetyClockReceiveTimestamp	Integer	[0..4398046511103]
vehicleSafetyClockTransmitTimestamp	Integer	[0..4398046511103]
checksum	Integer	[0..4294967295]
safeVehicleTypeConfirmation	Sequence	
vehicleType	IA5String	[1..32]
safetyProfile	IA5String	[1..32]
checksum	Integer	[0..4294967295]
vehicleError	Sequence	
time	Integer	[0..4398046511103]
vehCode	Enumerated	[0..4]
customCode	Integer	[0..255]
description	IA5String	[1..200]
vehicleSafetyFeedback	SequenceOf	[1..20]
VehicleSafetyFeedbackContainer	Sequence	
remainingTimeToStartBraking	Integer	[-32768..32767]
safetyViolations	SequenceOf	[0..5]
SafetyViolationsEnum	Enumerated	[0..12]
currentVehicleSafetyClockTime	Integer	[0..4398046511103]
vehicleProperties	Sequence	
basicVehicleClass	Enumerated	[0..15]
vehicleLength	Integer	[-524288..524287]
vehicleWheelbase	Integer	[-524288..524287]
vehicleRearOverhang	Integer	[-524288..524287]
vehicleWidth	Integer	[-524288..524287]
vehicleTireWidth	Integer	[-524288..524287]
vehicleTrackWidth	Integer	[-524288..524287]
vehicleMass	Integer	[1..1024]
vehicleSpeedLimit	Integer	[-16383..16383]
vehicleCuvatureLimit	Integer	[-32768..32767]
vehicleMaxAngularSteeringRate	Integer	[-32767..32766]

Figure 31: MVM message structure (part 2: optional containers)

The containers defined within this message are shown in Figure 32. With regards to the initial two containers for *header* and *e2eProtection* it corresponds to the MIM message structure in clause 7.1. The mandatory header container and its data elements are defined in clause C.1.

The ASN.1 representation of the Facilities layer message MVM shall follow the specifications as defined in Recommendation ITU-T X.680 [1].

Figure 32: Containers defined within the MVM message

8.2 e2eProtection (container: AvmE2EProtection)

8.2.1 End to end protection of the *MVM* container content

The structure of the `e2eProtection` container is portrayed in Figure 33 and shows the data elements supporting e2e protection mechanisms for the *MVM* container. These data elements share the same characteristics to those within the *MIM* container described in clause 7.2. An example how to calculate e2e protection data elements length and CRC32 is provided in the clause D.3.

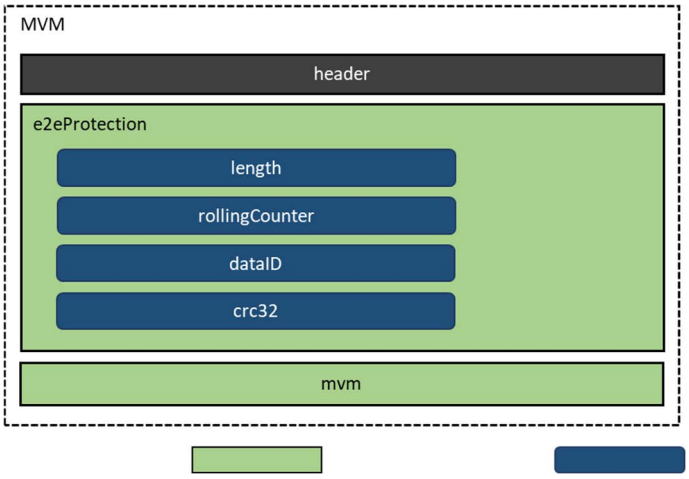


Figure 33: Structure of the *`e2eProtection`* container within the MVM message

8.2.2 *length* data element

The data element *length* represents the overall size in octets over that the protection mechanism is applied. The `ItsPduHeader` length (6 bytes) are subtracted from the complete length of the MVM.

8.2.3 *rollingCounter* data element

The data element *rollingCounter* is a recurring identifier of a MVM message. For the first transmission request the counter shall be initialize with 0 and shall be incremented by 1 for every subsequent send request of a MVM message. In case of overflow due to the size limit of the data element (0xFFFF) the counter shall restart with 0 for the next send request.

On the RO side, the evaluation of the counter of the received MVM against the counter of the previous MVM allows detection of the following conditions:

- Repetition: if the counters are the same.
- Communication error: if the counter increments is above a certain limit implying too many data lost.
- Normal condition: if the increment is by one or more than one but within the tolerated limits.

8.2.4 *dataID* data element

The data element *dataID* supports the verification of each transmitted MVM message at the SV side. *dataID* is to the best effort unique within the network of communicating systems as defined in [i.2].

In the RO subsystem, the comparison of the *dataID* of the received MVM against the expected *dataID* allows detection of the following conditions:

- Insertion of information: to verify that no additional information is added.
- Masquerading: to verify that no information is blanked out.
- Incorrect addressing: to verify whether an unknown *dataID* is transmitted.

8.2.5 *crc32* data element

As defined in [i.2], the Cyclic Redundancy Check is used in the SV subsystem to determine if bits flipped during the transmission of the MVM and thus the MVM has to be regarded as corrupted.

The calculation of CRC data element applies to the boundaries of the MVM bit stream while the starting 6 bytes from the `ItsPduHeader`are ignored. If necessary, the bit stream shall be enhanced with padding bits to result in an octet stream.

The CRC used in AUTOSAR E2E Profile 4 is a 32 bits CRC as described in [i.3].

8.3 *General Structure of* mvm (container: Mvm)

8.3.1 Overview

The *`mvm`* container within the MVM message starts right after the *`e2eProtection`* container as defined in the previous clause. The structure of a single `Mvm` container is portrayed in Figure 32.

It provides the following containers to transmit information from the VO subsystem to the RO subsystem:

1) `mvmDataControlField`
2) `systemManagementData`
3) `vehicleState`
4) `localizedPose`
5) `vidResponse`
6) `safetyTimeSyncResponse`

7) `vehicleError`

8) `vehicleSafetyFeedback`

9) `vehicleProperties`

All of these containers are marked as optional in order to achieve a high flexibility in future implementation profiles.

8.3.2 mvmDataControlField (optional container)

8.3.2.1 Introduction

The `mvmDataControlField` container is portrayed in Figure 34. It consists of a series of data elements that serve mainly for the purpose of control the bidirectional communication between the vehicles and the RO. This is equally available in the MVM structure.

Contained data elements are:

1) *mvmGenerationTime*

2) *rollingCounterFromMim as a sequence of RollingCounter*

3) *proprietaryExtensionField*

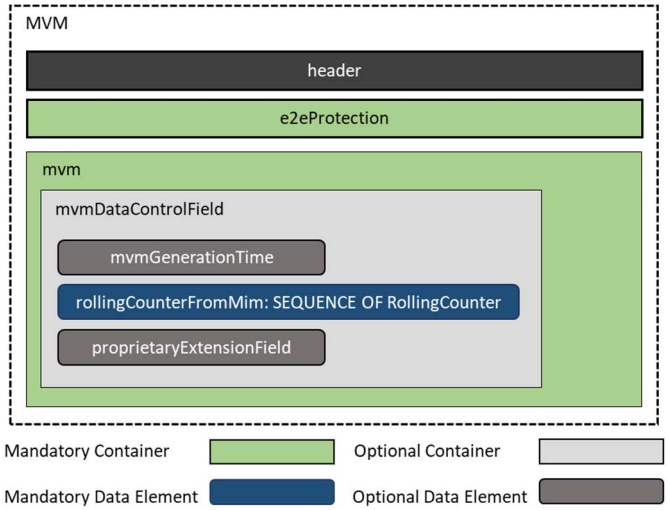


Figure 34: Structure of the `mvmDataControlField` container within an MVM

8.3.2.2 *mvmGenerationTime* data element

The *mvmGenerationTime* data element is optionally. It is used for check on the data freshness on the receive side from message generation time included as part of the message transmission from the transmitter side.

The format of the data element complies with CDD definition of *TimeStampIts* (ETSI ITS).

8.3.2.3 rollingCounterFromMIM (sequence)

The data element *rollingCounterFromMIM* serves as a mirror of the rolling counters that were received with the latest, up to 10 MIM messages from the RO. It enables the RO to recognize whether the vehicle system has received fresh data from the vehicle or if there are lost messages in the radio transmission.

8.3.2.4 *proprietaryExtensionField* (optional element)

This data element defines optionally 2 bytes are used to carry specific information or request from a specific SV to the RO, e.g. an information that the vehicle has entered a car wash station or opened the charging flap at an electric charging station.

The bytes are unformatted, i.e. the format is only defined between the carmaker and the infrastructure provider in order to provide additional specific information in the driving protocol.

8.3.3 systemManagementData (optional container)

8.3.3.1 Introduction

The `systemManagementData` container is portrayed in Figure 35. It consists of a series of optional data elements that serve for the purpose to exchange the identification labels of the bidirectional communication between the SVs and the RO. This is equally available in the MIM structure.

Contained data elements are:

1) *sessionID*
2) *missionID*
3) *vehicleID*
4) *facilityID*

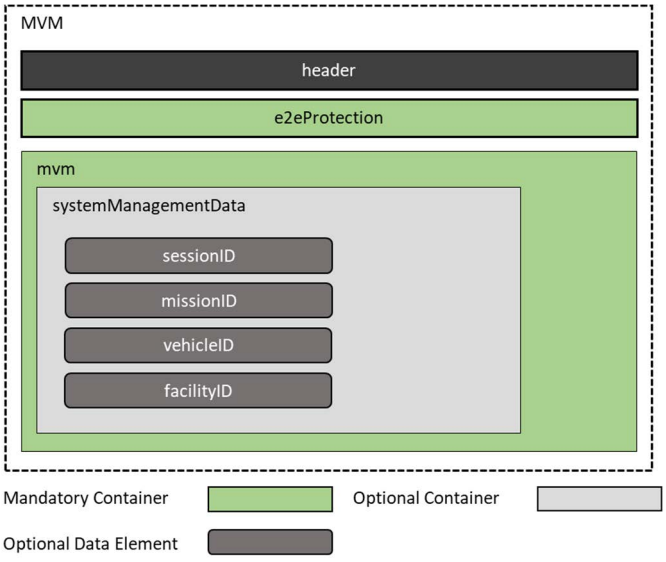


Figure 35: Structure of the `systemManagement` container within an MVM

8.3.3.2 *sessionID* data element

The vehicle confirms the according *sessionID* of a MIM with this data element. The RO identifies herewith the response from the associated vehicle. Clause 7.4.3.2 explains that the *sessionID* is negotiated between the RO and vehicle systems outside the driving protocol.

8.3.3.3 *missionID* data element

The *missionID* data element identifies a task that is negotiated and agreed upfront on both sides, the RO and the vehicle subordinated to an according *sessionID* date element. In MVM it has to comply with the current *missionID* data element of MIM.

8.3.3.4 *vehicleID* data element

The *vehicleID* data element is an optional identifier. The vehicle needs to know its own *vehicleID*, and this will be independent from *sessionID* values. The use of this data field corresponds to clause 7.4.3.4 in the MIM description and will be a mirror of the according *vehicleID* data element in MVM.

8.3.3.5 *facilityID* data element

The data element provides the option for the identification of the infrastructure facility, e.g. a parking structure or a logistics depot, depending on the use case. It serves for documentation purposes and should be identical in the corresponding MIMs and MVMs.

8.3.4 vehicleState (optional container: VehicleState)

8.3.4.1 Introduction

The `vehicleState` container carries signals about the actual status of the vehicle's condition with reference to the driving task. This return signals from the vehicle is usable in the RO subsystem for correction and refinement of the vehicle motion control VMC. The structure of the `vehicleState` container within the MVM message is shown in Figure 36.

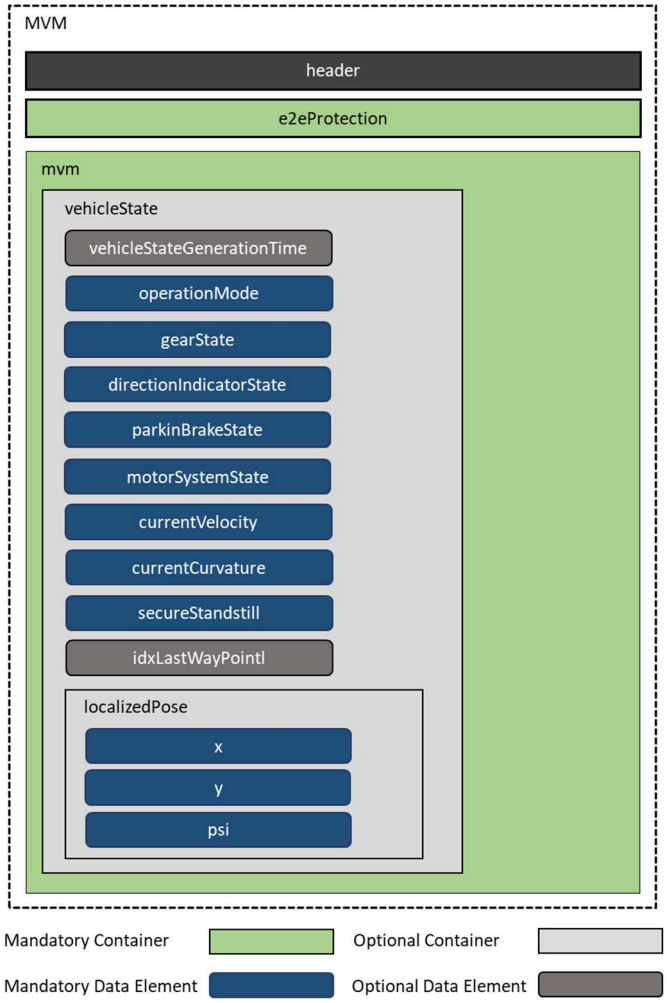


Figure 36: Structure of the `vehicleState` container within the MVM message

8.3.4.2 vehicleStateGenerationTime data element (optional)

This data element will be used for checks on the Data Freshness on receive side from Vehicle State Message generation from transmission point of View as Vehicle State Generation Time could originate from different module than radio transmitter module. The format of the data element complies with CDD definition of *TimeStampIts* [i.11].

8.3.4.3 *operationMode* data element

The data element *operationMode* comprises the modes described in Table 13 in response to the data element *driveCommandAction* within the MIM message.

Table 13: *operationMode* values

State	Description
unknown	default value
initializing	The vehicle is preparing for the mission but has not entered automated mode yet.
prepared	The vehicle is in automated mode but currently does not follow one of the control interfaces.
driving	The vehicle is in automated mode and actively follows one of the control interfaces (waypoints direct control, etc.). It has not reached the end of the given path yet. Also applies if the vehicle stopped temporarily.
terminating	The vehicle left automated mode and is terminating related functions.
suspend	The vehicle is in a critical error state and requires external operator intervention.
tempError	The vehicle is in a non-critical error state and is initiating a deceleration into stop and hold, prior to suspend.
humanInControl	Manual control of the vehicle has been taken over.
stationHold	The external mfg./customer environment interlocks have taken over and is holding the vehicle from marshalling.

8.3.4.4 *gearState* data element

The data element *gearSelected* reports in analogy to the gear request in clause 7.4.7.4 the states described in Table 14.

Table 14: *gearState* values

State	Description
park	P: Vehicle shall engage P.
backwards	R: Vehicle shall drive backwards.
neutral	N: Vehicle shall engage neutral gear.
forwards	D: Vehicle shall drive forwards.
unknown	vehicle gear state is unknown (not used as a command).

8.3.4.5 directionIndicatorState data element

The data element *directionIndicatorState* reports in analogy to the indicator request in clause 7.4.7.5 the states described in Table 15.

Table 15: *gearSelected* states

State	Description
off	The vehicle does not flash indicator lights at this stage.
right	Right indicator lights flashing.
left	Left Indicator lights flashing.
both	Right and left indicator lights flashing.
unknown	Flashing status is unknown (not used as a command).

8.3.4.6 parkingBrakeState data element

The data element *parkingBrakeState* reports in analogy to the parking brake request in clause 7.4.7.6 the states described in Table 16.

Table 16: *parkingBrakeState values*

State	Description
unknown	The status of the electric parking brake is unknown.
engaging	The parking brake is processing to become engaged.
engaged	The parking brake is fully engaged.
disengaging	The parking brake is processing to become disengaged.
disengaged	The parking brake is fully disengaged.

8.3.4.7 motorSystemState data element

The data element *motorSystemState* reports in analogy to the motor system request in clause 7.4.7.7 the states described in Table 17.

Table 17: *motorSystemState values*

State	Description
off	Propulsion Motor off.
on	Propulsion Motor on.
unknown	The propulsion motor status is unknown (not used as a command).

8.3.4.8 currentVelocity data element

The DE reports the current speed of the vehicle at the related local time stamp.

8.3.4.9 currentCurvature data element

The DE reports the current curvature of the vehicle at the related local time stamp.

8.3.4.10 secureStandstill data element

This is a Boolean data element reports a secured vehicle standstill, also on inclined roads. A secure standstill of the vehicle needs individually to be defined by the two partners of RO and VO subsystems and possibly includes the engagement of an electronic parking brake.

8.3.4.11 idxLastWayPoint data element

The optional DE is a 2-byte value. It is referenced to the path snippet concept in clauses 7.6 and 7.6.3.

The vehicle herewith reports to the RO subsystem index of the current `wayPoint` that has been passed in the array of consecutive `wayPoints`.

8.3.4.12 localizedPose (container: Pose)

The AVM system permanently senses the vehicle's position in a relative two-dimensional coordinate system (X and Y). In this optional container the vehicle reports its *x*, *y* position and heading *psi* with regards to the latest RO position and the delta movement calculated by vehicle odometry data.

8.3.4.13 *x* data element

This data element is 20 bit for the x position of the way point. Resolution is 1 cm.

8.3.4.14 *y* data element

This data element is 20 bit for the y position of the way point. Resolution is 1 cm.

8.3.4.15 *psi* data element

This data element is 16 bit for the desired heading position of the vehicle in that specific way point. A heading (*psi*) value of 0 represents a drive in x-direction. The unit is in 0,0001 radian and counted counter-clockwise.

8.3.5 vidResponse (optional container: VidResponse)

8.3.5.1 Introduction

With reference to clause 7.4.4.2, a seed can be negotiated using a Diffie-Hellman key exchange. This key exchange process allows the two participants to agree on a secret seed, even if everyone can read the communication.

NOTE: It is assumed that the communication between the participants is authenticated (payloads are signed). The proposed key exchange is for safety only. It does not contribute to security.

The structure of the `vidResponse` container is shown in Figure 37.

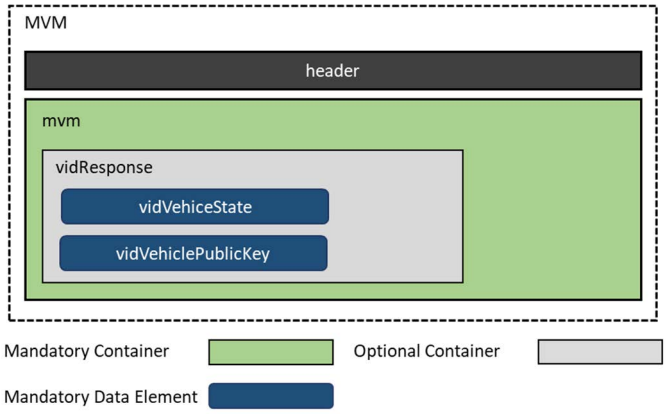


Figure 37: Structure of the `vidResponse` container

8.3.5.2 vidVehicleState data element

The data element *vidVehiclestate* describes the current state in the vehicle identification process and complies with the enumerated values shown in Table 18.

Table 18: *vidVehicleState* values

State	Description
undefined	Default value.
ready	Vehicle is ready to get flashing code.
lightFlashingInProgress	For the infrastructure to know that the vehicle is actually flashing its code.
lightFlashingCompleted	Flashing is finished.
lightFlashingFailed	Indicating that vehicle cannot flash because of vehicle error.
authorized	Vehicle identification was successful, and vehicle has switched its state.

8.3.5.3 VidVehiclePublicKey

The data element contains a 4-byte value for a public key from the vehicle to derive the vehicle identification secret.

8.3.6 safetyTimeSyncResponse (optional container: SafetyTimeSyncResponse)

8.3.6.1 Introduction

To calculate the ITS timestamp given in the *expirationTime* data element from the `drivingPermission` container in the MIM message, the RO subsystem needs to be able to continuously determine the current time in which the VO subsystem in an SV operates. This means that the RO determines the offset between the RO clock and the VO clock of an SV. Based on this offset, it is able to determine the time of the VO subsystem in an SV. The structure of the `safetyTimeSyncResponse` container is shown in Figure 38.

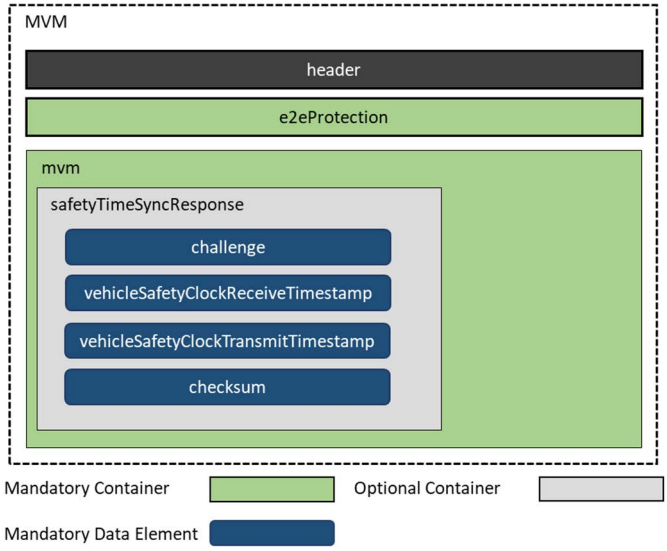


Figure 38: Structure of the `safetyTimeSyncResponse` container within the MVM message

8.3.6.2 *challenge* data element

The data element reflects the challenge as received in the `safetyTimeSyncRequest` container within the MIM message.

8.3.6.3 *vehicleSafetyClockReceiveTimestamp* data element

This clock data reflects the time of the RO subsystem when the `safetyTimeSyncRequest` (clause 7.4.6) was sent.

8.3.6.4 *vehicleSafetyClockTransmitTimestamp* data element

This clock data reflects the time in which the VO subsystem in an SV operates, adequately to currentVehicleSafetyClockTime, see clause 8.3.9.3.

8.3.6.5 *checksum* data element

This safety time synchronization shall be protected by a dedicated safety checksum. An example how to calculate data element checksum is provided in clause D.4.2.

8.3.7 safeVehicleTypeConfirmation (container: SafeVehicleTypeConfirmation)

8.3.7.1 Introduction

This safety-relevant container, which structure is shown in Figure 39, describes the vehicle type identifier. Though the vehicle type identifier was already sent to the RO by the backends, this is a safe confirmation by the SV.

The SV shall send the `safeVehicleTypeConfirmation` container once as soon as the vehicle identification was successful.

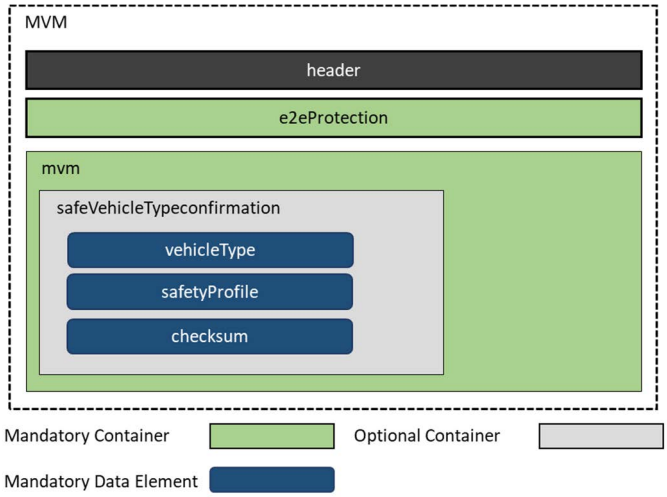


Figure 39: Structure of the `safeVehicleTypeConfirmation` container within the MVM message

8.3.7.2 *vehicleType* data element

The Vehicle Type Identifier is a string, agreed upon between OEM and the Remote Vehicle Operation provider.

From the Vehicle Type Identifier, the vehicle-specific properties e.g. wheelbase, front and rear overhangs, etc., can be inferred. Operator Backend (OB) and RO subsystem already received the Vehicle Type Identifier from the VB when the session was initiated.

8.3.7.3 *safetyProfile* data element

This data element carries a string information that is exchanged beyond the VMC protocol. The safety profile is optionally negotiated via an OB and a VB. For confirmation of the backend exchange this element supports safety mechanism of the `safeVehicleTypeConfirmation` container.

8.3.7.4 *checksum* data element

This safety time synchronization needs to be protected by a dedicated safety checksum. An example how to calculate data element checksum is provided in clause D.4.3.

8.3.8 vehicleError (container: VehicleError)

8.3.8.1 Overview of the *vehicleError* container

This container is an optional data carrier and is added to the MVM message in case an error appears during VMC of the vehicle. Figure 40 illustrates the data elements in the `vehicleError` container.

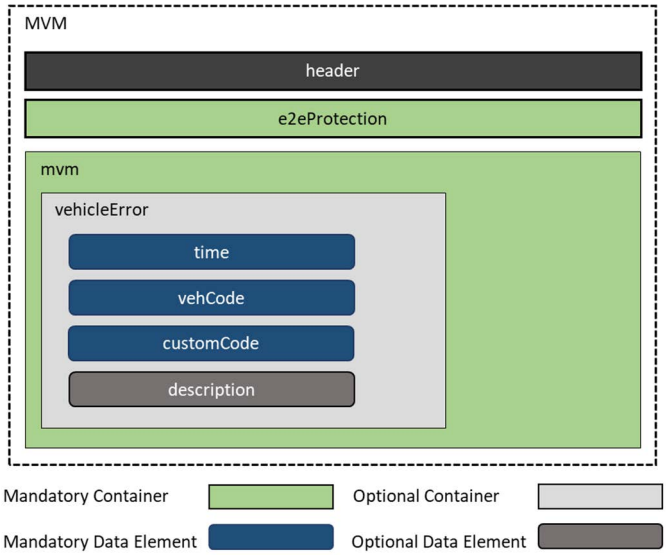


Figure 40: Structure of the `vehicleError` container within the MVM message

8.3.8.2 *time* data element

The data element shows the timestamp when the error occurred.

8.3.8.3 *vehCode* data element

Depending on the given error by the vehicle code data element *vehCode*, the infrastructure either tries to resolve the issue or aborts the mission. It saves this state for logging purposes and forwards its content to the backend. A list of mapped errors is presented in Table 19.

Table 19: *vehCode* error codes

State	Description
unspecified	Any kind of error that is not specified otherwise. Infrastructure aborts the mission.
pathNotDriveable	The vehicle can not follow the given waypoints based on the given detectedVehiclePose. Infrastructure tries to plan a different path or aborts the mission otherwise.
onboardVehicleFault	Failure during vehicle onboarding.
communicationFault	Vehicle internal communication error.
vehicleEgressFault	Failure during vehicle AVM egress or shutdown process.

8.3.8.4 *customCode* data element

It contains a data element for a customer specific error code. The infrastructure will not further interpret this value.

8.3.8.5 *description* data element (optional)

This data element contains up to 200 characters. It carries a description (verbal or coded) with further details about the error.

8.3.9 *vehicleSafetyFeedback* (optional sequence)

8.3.9.1 Overview

This defines optionally a sequence of up to up to 20 *VehicleSafetyFeedback* containers within one MVM message. It helps to identify the safety feedback information of several sub-loops that can happen during one MVM information cycle.

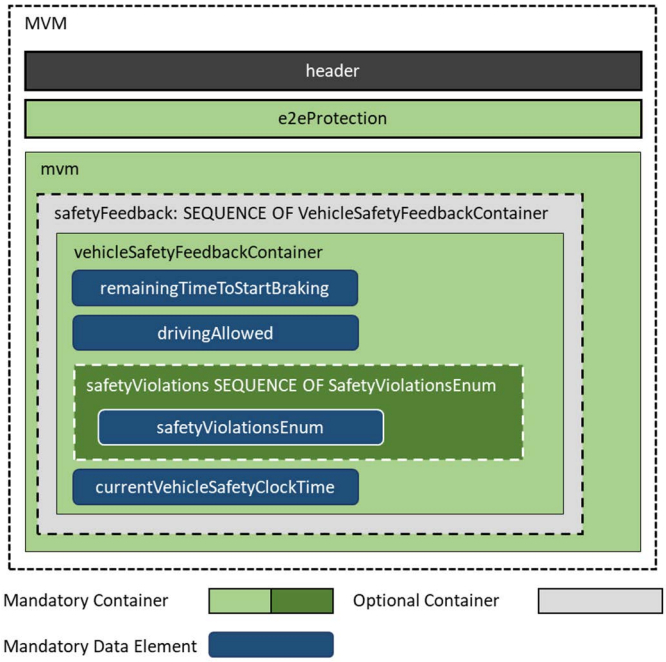


Figure 41: Structure of the `vehicleSafetyFeedback` container within the MVM message

8.3.9.2 VehicleSafetyFeedbackContainer (container)

8.3.9.2.1 Introduction

This container hosts the safety feedback information of a specific safety cycle.

8.3.9.2.2 *remainingTimeToStartBraking* data element

The time which the vehicle is allowed to keep driving until brakes shall be engaged.

8.3.9.2.3 safetyViolations: SEQUENCE OF SafetyViolationsEnum

This reports a sequence of up to 5 safety violations from the DE *safetyViolationsEnum* due to the specific safety cycle which as a result leads to stopping the vehicle. This structure covers the fact that during one safety cycle more than one stopping reason can occur.

8.3.9.2.4 *safetyViolationsEnum data* element

Table 20 presents a list of enumerators available to support reporting a safety violation as described in clause 8.3.9.2.3.

Table 20: List of *safetyViolationsEnum* enumerators

State	Description
noViolation	No violation
noDrivingPermissionReceived	No driving permission received
lastDrivingPermissionTooOld	Last driving permission too old
crcViolationClockSyncRequest	CRC violation clock sync request
crcViolationDrivingPermission	CRC violation driving permission
expirationTimeViolation	Expiration time violation
drivingDirectionMismatch	Driving direction mismatch
velocityViolation	Velocity violation
curvatureMinViolation	Curvature min violation
curvatureMaxViolation	Curvature max violation
expirationTimeTooHigh	Expiration time too high
monitoring	Monitoring

8.3.9.3 *currentVehicleSafetyClockTime* data element

The DE contains the time stamp of the specific safety loop in the vehicle as described in detail with the specific *vehicleSafetyFeedback* container.

8.3.10 vehicleProperties (container: VehicleProperties)

8.3.10.1 Introduction

This container is an optional data carrier and is added to the MVM message mainly as a surrogate for missing static vehicle data that AVM system will possibly need. In specific use cases the static vehicle data will be transferred upfront to a VMC protocol e.g. via backend communication.

8.3.10.2 Overview of the *vehicleProperties* container

The data elements structure of the `vehicleProperties` container is shown in Figure 42.

Figure 42: Structure of the *`vehicleProperties`* container within the MVM message

8.3.10.3 *basicVehicleClass* data element

This data element specifies a list of generic vehicle classes that the AVM system will cope with. The list of defined vehicle classes as enumerators is shown in Table 21.

Table 21: List of *basicVehicleClass* enumerators

State	Description
none	Not equipped, not known or unavailable
unknown	Does not fit any other category
special	Special use
moto	Motorcycle
car	Passenger car
car-Other	Four tire single units
bus	Passenger busses
axleCnt2	Two axle, six tire single units
axleCnt3	Three axle, single units
axleCnt4	Four or more axle, single unit
axleCnt4Trailer	Four or less axle, single trailer
axleCnt5Trailer	Five or less axle, single trailer
axleCnt6Trailer	Six or more axle, single trailer
axleCnt5MultiTrailer	Five or less axle, multi trailer
axleCnt6MultiTrailer	Six axle, multi trailer
axleCnt7MultiTrailer	Seven or more axle, multi trailer

8.3.10.4 *vehicleLength* data element

This data element *vehicleLength* contains the overall vehicle length in (cm).

8.3.10.5 *vehicleWheelbase* data element

The data element *vehicleWheelbase* contains the wheelbase value (cm).

8.3.10.6 *vehicleRearOverhang* data element

This data element contains the rear overhang value (cm).

8.3.10.7 *vehicleWidth* data element

The data element *vehicleWidth* contains the overall vehicle width value (cm).

8.3.10.8 *vehicleTireWitdth* data element

The data element *vehicleTireWidth* contains the specified width of tires value. It is applied for precise control if the vehicle drives over wheel chocks and ramps.

8.3.10.9 *vehicleMass* data element

The *vehicleMass* data element represents a rough vehicle mass value, it is taken from the CDD.

8.3.10.10 *vehicleTrackWidth* data element

This data element contains the vehicle track width value (cm) which is applicable for precise control if the vehicle drives over wheel chocks and ramps.

8.3.10.11 *vehicleSpeedLimit* data element

The data element *vehicleSpeedLimit* contains the limit value of the vehicle speed according to the use case, unit in (cm/s).

8.3.10.12 *vehicleCurvatureLimit* data element

The data element *vehicleCurvatureLimit* contains the limit value of the vehicle curvature according to the use case, symmetrically applicable for left and right turns.

8.3.10.13 *vehicleMaxAngularSteeringRate* data element

This data element reports the static value of a maximum steering rate for the according use case, with reference to the wheel angle, not steering wheel angle. Unit is specified as rad/s with a resolution of 0,0001.

Annex A (normative): ASN.1 modules

This clause provides the normative ASN.1 modules containing the syntactical definitions of the MIM (Table A.1) and MVM (Table A.2) PDUs as well as the specification of the components shared between MIM and MVM (Table A.3).

Table A.1: MIM-PDU-Descriptions ASN.1 module information

Module name	MIM-PDU-Desriptions
OID	{itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103882) mim (6) major-version-1 (1) minor-version-1 (1)}
Link	https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/raw/v2.1.1/MIM-PDU-Descriptions.asn
SHA-256 hash	f6a9e375dfbab19cf12871d11d47186fdf7b724dc338e36219eaaf986b3a0f04

Table A.2: MVM-PDU-Descriptions ASN.1 module information

Module name	MVM-PDU-Desriptions
OID	{itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103882) mvm (4) major-version-1 (1) minor-version-1 (1)}
Link	https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/raw/v2.1.1/MVM-PDU-Descriptions.asn
SHA-256 hash	32b919f1b4f2512e5c6b1153c9cd81c60109a40a9f7227d0ecb3c79c9bd9df1d

Table A.3: AVM-Commons ASN.1 module information

Module name	AVM-Commons
OID	{itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (103882) avmCommons (5) major-version-1 (1) minor-version-1 (1)}
Link	https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/raw/v2.1.1/AVM-Commons.asn
SHA-256 hash	42ddc1f5ba86e9ccbb4895b1df71e7deef72093abde8e94327e5ac8a6c244841

Annex B (informative): AVM modules in readable form

The readable format of the AVM Commons content, and the MIM and MVM PDUs components is available at the following URL:

- https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/tree/release2/docs.

Table B.1: Readable format description of the AVM-Service

Filename	File URL
MIM-PDU-Descriptions.md	https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/blob/v2.1.1/docs/MIM-PDU-Descriptions.md
MVM-PDU-Descriptions.md	https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/blob/v2.1.1/docs/MVM-PDU-Descriptions.md
AVM_Commons.md	https://forge.etsi.org/rep/ITS/asn1/avp_ts103882/-/blob/v2.1.1/docs/AVM-Commons.md

Annex C (normative): MIM and MVM guidelines

C.1 Message payload encapsulation

It is recommended to support payload encapsulation by means of the Data Frame (DF) `ItsPduHeader` as defined in ETSI TS 102 894-2 [i.11]. The `ItsPduHeader` includes:

- *protocolVersion*: Version of the ITS payload contained in the message.
- *messageId*: Type of the ITS payload contained in the message.
- *stationId*: Identifier of the ITS-S that generated the message.

The *protocolVersion* rules are explained in clause C.3.

The *messageId* for MIM and MVM are defined in ETSI TS 102 894-2 [i.11].

NOTE: The ASN.1 code in normative Annex A uses the `ItsPduHeader` for payload encapsulation making it normative for the present document.

C.2 Message encoding scheme

ASN.1 is associated with many encoding rules. It is recommended to use the Unaligned Packing Encoding Rules (UPER) as specified in Recommendation ITU-T X.691 [i.4] for the encoding of MIM and MVM messages.

C.3 Protocol version handling

The value of the protocol version for the messages MIM and MVM is individual but not independent of each other. Therefore, a DE for protocol versioning shall be defined in the payload header for MIM and MVM. If recommendation in clause C.1 is followed, the value of the data element *protocolVersion* of the messages MIM and MVM shall match.

To maintain the versioning of the protocol in ASN.1, it is recommended to follow the versioning guideline defined in ETSI TS 102 894-2 [i.11].

C.4 MIM and MVM configuration parameter values

A list of the required system parameters specified in clause 6.2 and their recommended values or value ranges is provided in Table C.1.

Table C.1: Configuration parameter values recommended for the MIM/MVM generation

Parameter	Type	Value	Description
T_GenMIM	Time (ms)	100	Generation time interval of MIM
T_EventMIM	Time (ms)	1 000	Time interval for event-driven transmissions of MIM
N_EventMIM	Number	5	Number of MIMs that can be generated during *T_EventMim*
T_GenMVM	Time (ms)	100	Generation time interval of MVM
T_EventMVM	Time (ms)	1 000	Time interval for event-driven transmissions of MVM
N_EventMVM	Number	5	Number of MVMs that can be generated during *T_EventMvm*

Annex D (normative): Functional Safety Elements

D.1 Introduction

The present document specifies the protocol to implement the VMC operation interface for AVM related use cases. An implementation of the VMC operation interface requires functional safety support due to ISO 26262 [i.9] and IEC 61508 [i.10] as addressed in ISO 23374-1 [i.6]. Therefore, the protocol is enhanced with elements that help implementing functional safety requirements.

At first hand this is noticeable with the `AvmE2Eprotection` container. Clause D.3 presents and exemplary calculation. The `AvmE2Eprotection` container was developed in compliance with [i.9] and [i.10] as an end-to-end data protection method. Another element is the checksum in the MIM vehicle container that can be used for specific OEM vehicles and is not further specified in the present document. In clause D.4 there are other safety checksum calculations explained that cover special content of optional containers.

D.2 Support to functional safety concepts

All elements can be handled flexibly to support functional safety requirements of the AVM service or related use cases in different system architectures, especially in given electronic vehicle architectures. It is provided in the VMC protocol on the facilities layer level and is independent of security mechanisms that are designed on lower layers. In general, they give implementers the freedom to design an end-to-end protection for integrity of signals that are relevant for functional safety as specified in ISO 26262 series [i.9] and IEC 61508-1 [i.10], while the endpoints in the respective architectures are flexibly chosen.

Figure D.1 depicts the signal path of safety relevant signals from and to the safety end points. Following the architecture considerations are listed that can be applied to AVM service.

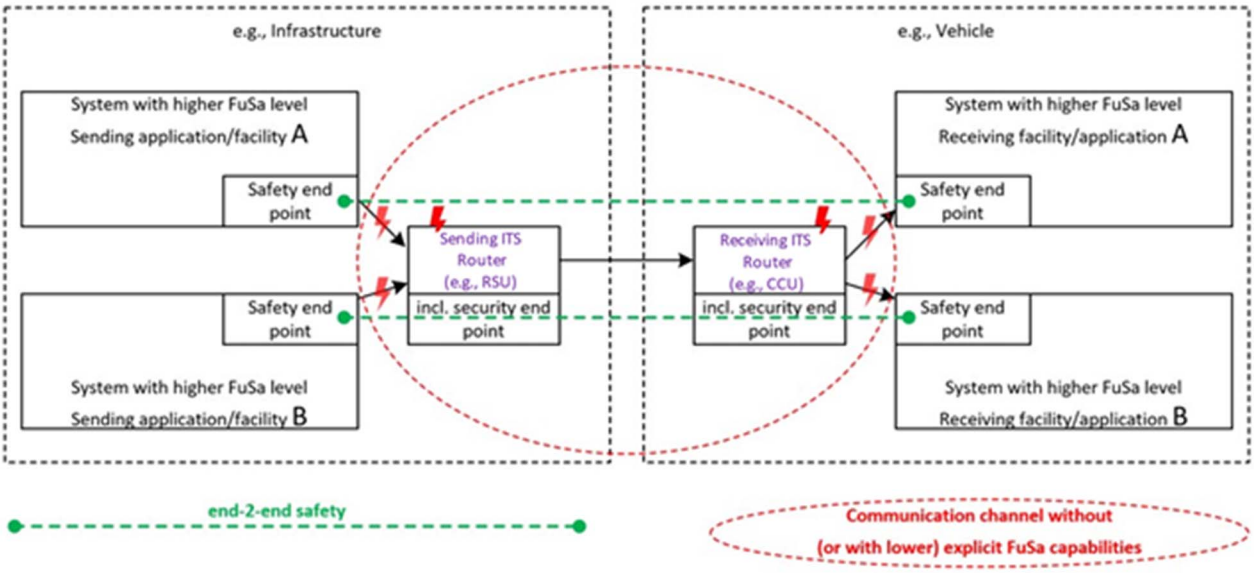


Figure D.1: Signal path to safety end points

- The ITS station is part of the vehicle, but it is in most cases distributed among different components of the vehicle, in the ITS Router (also called "Communication and Control Unit" (CCU) potentially using different communication technologies, e.g. short-range broadcast communication, cellular network communication, etc.) and in multiple different hardware units.
- The architecture of the implementation and the specific distribution of the V2X functionality depend strongly on the manufacturer of the vehicle, on the vehicle type, specific constraints, design decisions and costs.

- Not all the components are developed following the safety standards and fulfilling all safety requirements. Since developing safe software and safe hardware is a complex and expensive task, the developers try to focus safety on as few components as possible and keep safety requirements away from other components. They prove that the overall system is safe, even if these parts of the vehicle are not developed in a safe manner, because there lies no safety load on these components.

- There might be different hardware/software components on infrastructure side and on vehicle side, fulfilling different levels of safety and requirements for the respective application A or B.

- Between those safe systems for A and B, there is a "not so safe" communication channel, simply not fulfilling all safety requirements of all the applications.

- There are ITS Routers, as shown in Figure 37, also not fulfilling all safety requirements, because they need to be cheaper, they are developed in a more generic way, or they cannot implement all safety requirements that a safe application could have.

- The safe components for A and B now consider themselves connected by a channel not fulfilling all the safety requirements, and this includes also the ITS Routers, as shown in Figure 37. From safety point of view, they do not fulfil the requirements and are part of an open channel not giving all the safety which the application A and B need. This is indicated by the red flash symbols, e.g. a bit flip can happen on that part of the signal path. So, the safety requirements are implemented and checked in the safe components for A and B.

- Terminology explanation: the security can be arranged "end-2-end" between ITS Routers, and the safety is done "end-2-end" between the safe systems implementing the applications A and B. In this way, "end-2-end" does not mean the same for security and for safety.

The protocol is offering two methods to support safe driving of remote-controlled vehicles:

- The Driving Permission concept referenced in ISO 23374-1 [i.6], 5GAA TR T-220002 [i.8], and described in the VDA AVP position paper [i.7]. This safety concept is possible to implement by means of the `drivingPermission` container described in clause 7.4.5.

- An Emergency Stop concept based on the data element *emergencyStop* residing in the `driveCommand` container described in clause 7.4.7.8.

Additionally, the implementation of a time-out criterion for the MIM message leads directly to an action equal to that given by the implementation of the *emergencyStop* safety concept.

D.3 End to End Protection calculation

D.3.1 Overview

The AVM entity messages provides the `AvmE2Eprotection` (clauses 7.2 and 8.2) as general end-to-end protection container. The implementation of the container follows the Autosar profile 4 specification given in [i.2] mandating the fields length, rollingCounter, dataID, and crc32. These fields are part of the `e2eProtection` container which DEs are described in clauses 7.2.2, 7.2.3, 7.2.4 and 7.2.5.

The purpose of this clause is to provide some guidelines to implementers about the computation and verification of the MIM and MVM end-to-end protection container *length* and *crc32* DEs. The clause implies that recommendations given in clauses C.1 and C.2 are followed.

The end-to-end protection container `AvmE2Eprotection` contains two DEs (*length* and *crc32*) whose values can only be computed after the `Mim` (respectively `Mvm`) containers have been encoded.

Using UPER encoding rules, and due to the ASN.1 definition of the *length* and *crc32* DEs, those two DEs will always be encoded over the same number of bits (16 bits and 32 bits respectively).

Using UPER encoding rules, and due to the ASN.1 definition of MIMs and MVMs, and of the first `ItsPduHeader` and `AvmE2Eprotection` data frames, the *length* and *crc32* DEs will always be found at the same location in the octet stream resulting from the MIM and MVM messages encoding.

The same applies to *rollingCounter* and *dataID* DEs.

The UPER-encoded `ItsPduHeader` data frame is 6 bytes long. Note that this data frame is not extensible as specified in Recommendation ITU-T X.691 [i.4].

Note also that neither `MIM` and `MVM` messages, and `AvmE2Eprotection` container are extensible. They shall remain so.

The 16 bits of the end-to-end protection container' *length* DE will be found right after the UPER-encoded `ItsPduHeader` data frames 6 bytes. Therefore, the *length* DE will be encoded over the 7th and 8th bytes, in network-byte order (i.e. big endian).

The 16 bits of the end-to-end protection container' *rollingCounter* DE will be found next. That is, the *rollingCounter* DE will be encoded over the 9th and 10th bytes, in network-byte order (i.e. big endian).

The 32 bits of the end-to-end protection container' *dataID* data element will be found next. In other words, the *dataID* DE will be encoded over the 11th to 14th bytes, in network-byte order (i.e. big endian).

The 32 bits of the end-to-end protection container' *crc32* DE will be found after skipping 6 additional bytes (encoding of the *rollingCounter* and *dataID* DEs), which means that the *crc32* data element will be encoded over the 15th to 18th bytes, in network-byte order (i.e. big endian).

It is recommended to use the following algorithm when sending (encoding) MIMs or MVMs:

Step 1: The MIM or MVM message is UPER-encoded. The *length*, *rollingCounter*, *dataID* and *crc32* data elements in the `AvmE2EProtection` container were uninitialized (e.g. they may be all set to 0).

As an alternative, the *rollingCounter* and *dataID* data elements may have been initialized to their appropriate values, before UPER-encoding, unless the AUTOSAR `E2E_P04Protect` function is invoked, see the note below. In that case, step 4 below is useless.

Step 2: The length is computed over the resulting MIM or MVM encoded message octet stream: the `ItsPduHeader` length (6 bytes) needs to be subtracted.

Step 3: The computed length is written in the MIM or MVM encoded message octet stream, over the 6th and 7th bytes, in network-byte order (i.e. big endian).

Step 4: The *rollingCounter* value is written in the MIM or MVM encoded message octet stream, over the 9th and 10th bytes, in network-byte order (i.e. big endian).

Step 5: The *dataID* value is written in the MIM or MVM encoded message octet stream, over the 11th to 14th bytes, in network-byte order (i.e. big endian).

Step 6: The checksum is computed over the resulting MIM or MVM encoded message octet stream, ignoring the first 6 bytes (`ItsPduHeader` encoding) and ignoring the checksum bytes (to align with ISO 23374-1 [i.6]).

Step 7: The computed checksum length is written in the MIM or MVM encoded message octet stream, over the 15th to 18th bytes, in network-byte order (i.e. big endian).

Step 2, step 3, step 4, step 5, step 6 and step 7 above may be achieved by calling the AUTOSAR `E2E_P04Protect` function as defined in [i.3]. The offset shall be set to $6 \times 8 = 48$ bits (to skip and ignore the `ItsPduHeader` encoding). This function will also set the *rollingCounter* and *dataID* data elements.

The following algorithm shall be followed when receiving a MIM or MVM message:

Step 1: The received message stream 15th to 18th bytes of are read to extract the end-to-end protection container' *crc32* data element value encoded in the message.

Step 2: The checksum is computed over the received octet stream, ignoring the first 6 bytes (`ItsPduHeader` encoding) and ignoring the checksum bytes (to align with [i.2]).

Step 3: The *crc32* DE value is compared with the computed checksum.

Step 4: The whole received message stream is UPER-decoded, and the end-to-end protection container' *length*, *rollingCounter* and *dataID* data elements can be verified as well.

Step 2 above may be achieved by calling the AUTOSAR `E2E_P04Check` function as defined in [i.3]. The offset shall be set to 6 × 8 = 48 bits (to skip and ignore the `ItsPduHeader` encoding).

D.3.2 MVM example

The following example represents a calculation guideline for implementers of the end-to-end protection container in MVM messages.

A dummy MVM message is used to illustrate the different steps, Its XML Encoding Rules (XER) are given here:

```
<MVM>
   <header>
      <protocolVersion>0</protocolVersion>
      <messageId>19</messageId>
      <stationId>2696004307</stationId>
   </header>
   <e2eProtection>
      <length>0</length>
      <rollingCounter>0</rollingCounter>
      <dataID>0</dataID>
      <crc32>0</crc32>
   </e2eProtection>
   <mvm>
      <systemManagementData>
         <sessionID>mysessionid202312081030</sessionID>
         <missionID>mymissionid103001200</missionID>
      </systemManagementData>
      <vehicleState>
         <vehicleStateGenerationTime>1940637616166</vehicleStateGenerationTime>
         <operationMode><driving /></operationMode>
         <gearState><forwards /></gearState>
         <directionIndicatorState><right /></directionIndicatorState>
         <parkingBrakeState><disengaged /></parkingBrakeState>
         <motorSystemState><on /></motorSystemState>
         <currentVelocity>13260</currentVelocity>
         <currentCurvature>-5000</currentCurvature>
         <secureStandstill><false /></secureStandstill>
         <idxLastWayPoint>369</idxLastWayPoint>
      </vehicleState>
      <vidResponse>
         <vidVehicleState><authorized /></vidVehicleState>
         <vidVehiclePublicKey>8024936506837307262</vidVehiclePublicKey>
      </vidResponse>
      <safeVehicleTypeConfirmation>
         <vehicleType>myvehicletype</vehicleType>
         <safetyProfile>mysafetyprofile</safetyProfile>
         <checksum>123456789</checksum>
      </safeVehicleTypeConfirmation>
   </mvm>
</MVM>
```

Step 1:

MVM message is UPER-encoded. The `length` and `rollingCounter` and `dataID` and `crc32` data elements were all set to 0:

```
0013A0B1 C2D30000 00000000 00000000 00003A31 B6F9E797 9F3D3BF7 69C8C983
266C5930 70C58336 07B7CEDD 3CF9E9DF BB4E462C 19B060C5 930619C3 D7040826
1B31E796 D8F00171 ADEBC9A7 856341EF CCDBE7B6 5D1A71EC CBD3CF0C AEDBE79E
1CD97A79 E1CB7E6D 3B3283AD E68A80
```

NOTE: In that example, there are seven padding bits at the end of the stream.

Step 2:

The length is computed as (111- 6) = 105 (0x00 69) bytes.

Step 3:

The length is written in the encoded octet stream:

```
0013A0B1 C2D30069 00000000 00000000 00003A31 B6F9E797 9F3D3BF7 69C8C983
266C5930 70C58336 07B7CEDD 3CF9E9DF BB4E462C 19B060C5 930619C3 D7040826
1B31E796 D8F00171 ADEBC9A7 856341EF CCDBE7B6 5D1A71EC CBD3CF0C AEDBE79E
1CD97A79 E1CB7E6D 3B3283AD E68A80
```

Step 4:

Assuming the rolling counter value should be 3521 (0x0D C1), it is written in the encoded octet stream:

```
0013A0B1 C2D30069 0DC10000 00000000 00003A31 B6F9E797 9F3D3BF7 69C8C983
266C5930 70C58336 07B7CEDD 3CF9E9DF BB4E462C 19B060C5 930619C3 D7040826
1B31E796 D8F00171 ADEBC9A7 856341EF CCDBE7B6 5D1A71EC CBD3CF0C AEDBE79E
1CD97A79 E1CB7E6D 3B3283AD E68A80
```

Step 5:

Assuming the data ID value should be 0xB304 87DC, it is written in the encoded octet stream:

```
0013A0B1 C2D30069 0DC1B304 87DC0000 00003A31 B6F9E797 9F3D3BF7 69C8C983
266C5930 70C58336 07B7CEDD 3CF9E9DF BB4E462C 19B060C5 930619C3 D7040826
1B31E796 D8F00171 ADEBC9A7 856341EF CCDBE7B6 5D1A71EC CBD3CF0C AEDBE79E
1CD97A79 E1CB7E6D 3B3283AD E68A80
```

Step 6:

The checksum is computed over (i.e. ignoring the UPER-encoded `ItsPduHeader` data frame 6 bytes, and the checksum placeholder):

```
0069 0DC1B304 87DC
3A31 B6F9E797 9F3D3BF7 69C8C983
266C5930 70C58336 07B7CEDD 3CF9E9DF BB4E462C 19B060C5 930619C3 D7040826
1B31E796 D8F00171 ADEBC9A7 856341EF CCDBE7B6 5D1A71EC CBD3CF0C AEDBE79E
1CD97A79 E1CB7E6D 3B3283AD E68A80
```

Computed checksum (using AUTOSAR E2E Profile 4 CRC polynomial): 0x8524A071.

Step 7:

The checksum is written in the encoded octet stream:

```
0013A0B1 C2D30069 0DC1B304 87DC8524 A0713A31 B6F9E797 9F3D3BF7 69C8C983
266C5930 70C58336 07B7CEDD 3CF9E9DF BB4E462C 19B060C5 930619C3 D7040826
1B31E796 D8F00171 ADEBC9A7 856341EF CCDBE7B6 5D1A71EC CBD3CF0C AEDBE79E
1CD97A79 E1CB7E6D 3B3283AD E68A80
```

D.4 Safety Checksum calculations

D.4.1 Overview

Several checksum data elements are described in the normative clauses of the present document. This clause provides examples on how to use the checksum data elements and provides their calculations as well for implementers.

The `AvmE2Eprotection` container provides a safety checksum in both messages MIM and MVM.

In addition there are checksum DEs provided in the following containers:

- `DrivingPermission`
- `SafetyTimeSyncRequest`
- `SafetyTimeSyncResponse`
- `SafeVehicleTypeConfirmation`

These containers carry an additional checksum to ensure that the payload was not modified accidentally and was generated for that specific SV.

D.4.2 SafetyTimeSyncRequest checksum

The following algorithm shall be used to calculate the checksum for the containers `SafetyTimeSyncRequest`, `SafetyTimeSyncResponse` and `SafeVehicleTypeConfirmation`:

```
checksum = CRC32K9(serializedData + (seed XOR TRANSFORMATION_CONSTANTA))
```

Checksum algorithm in detail:

1) Serialize the message payload as specified in *AVM-message-protocol*, by concatenating the byte-representation of all signals (except the checksum signal).

2) Append the byte-representation of the transformed vehicle-id seed, which is calculated as

```
transformedVehicleIdSeed = seed XOR TRANSFORMATION_CONSTANT
```

3) Calculate the CRC-32K/9 checksum over the data generated in step 1. and 2.

Seed refers to VidRequest.seed.

The whole 64 bit of the transformation constant *`TRANSFORMATION_CONSTANT`* shall only be known to components that are compliant to an Automotive Safety Integrity Level (ASIL) rating.

D.4.3 DrivingPermission checksum

The following algorithm shall be used to calculate the checksum for the container `DrivingPermission`:

```
checksumDrivingPermission = CRC32K9(serializedData + (seed XOR
TRANSFORMATION_CONSTANTA)) XOR TRANSFORMATION_CONSTANTB
```

seed refers to *VidRequest.seed*

Checksum algorithm in detail:

This algorithm equals the checksum calculation described in previous clause, with an additional XOR transformation applied to the result.

Since the Vehicle Identification code needs to be known for calculating the checksum, safety relevant payloads cannot be exchanged unless the Vehicle Identification process was successful.

Annex E (informative): AVM Entity guidelines

E.1 Introduction

An implementation of the AVM entity supports Automated Vehicle Marshalling when the SV is under System Management [i.6], in other words, the AVM system has authority over the SV as shown in the state transition diagram in Figure E.1.

E.2 AVM state transition diagram

ISO-233741 [i.6] introduces a State Transition Diagram (STD) to map the condition of an SV in an Automated Valet Parking (AVP) flow. Figure E.1 shows and illustrative STD from the RO perspective. The STD is an exemplary use of MIMs through a VMC operation interface (clause 4.3.3) that allows the SO to request a condition of the SV using the `driveCommand` container (clause 7.4.7) and a selected `controlInterface` (clause 7.5). It is recommended to follow the STD in [i.6] to support an implementation of the VMC operation interface.

The `driveCommand` enables transitioning between states within System Authority and into or out of the *Normal* superstate within Automated Vehicle Operation. The request from RO is driven using the *driveCommandAction* DEs in the `driveCommand` container. When the SV condition is *Normal*, the transition between the states *Drive* and *Pause* is driven by the selected VMC methodology (clause 4.3.4) chosen by means of `controlInterface`. Transitioning from *Drive* to *Pause* implies using the selected VMC methodology to bring the SV to a standstill and maintain a stationary condition. Similarly, transitioning from *Pause* to *Drive* implies resuming the manoeuvring of the SV along a path or trajectory. Using the VMC methodology is the normal operation for manoeuvring an SV.

ISO-233741 [i.6] defines two requirements that are adopted as recommendation in the present document:

- That an SV transitions into System Authority implies that a session has been stablished and that clause 4.3.2 is fulfilled.
- When an SV transitions into Automated Vehicle Operation implies that a mission has been assigned.

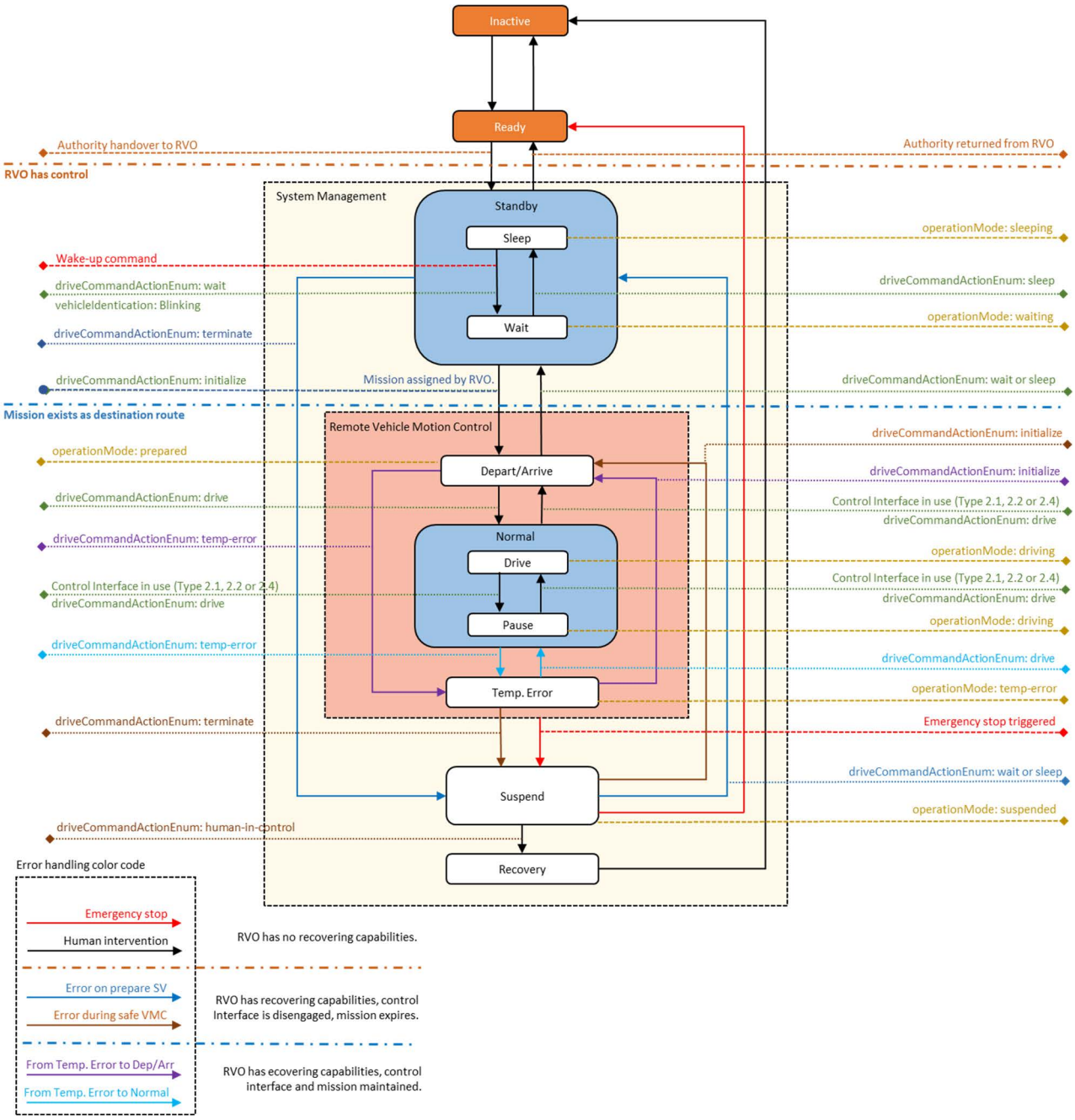


Figure E.1: ISO-233741 State Transition Diagram in the context of AVM

E.3 General functionality of the MIM and MVM messages

The MIM and MVM messages are intended to be used for AVM when the SV finds itself under System Authority as specified in clause 4.3.2. This means that the SV is awake and/or ready to receive the AVM entity messages and react accordingly. These conditions are fulfilled when the SV is in a *wait* condition. The required wake-up command shown in Figure E.1 transitions the SV into the *wait* condition and it is expected that this command is given to the SV using a management interface between VB and VO as illustrated in Figure 1.

MIM and MVM messages are generated according to clause 6.2 and the composition of the messages can vary each time a message is sent with containers and DEs that are optional being taken in an out. The MVM message is a response message of the VO to a previously MIM message sent by the RO. However, both messages are disseminated independently by the VO and the RO subsystems, thus they do not require acknowledgement or a fixed response interval but to meet the specifications given in clause 6.

The structure of MIMs and MVMs leverages the ASN.1 flexibility to provide a communication protocol with enough robustness to support diverse AVM use cases. The content of the AVM entity messages is to be tailored accordingly by implementers to support the AVM functionality and use case of interest. To easier tailoring MIMs and MVMs, DEs are containerized based on the functionality they support.

NOTE: If a mandatory DE is not used it stays constant at a default value.

E.4 IP based unicast communication and security

E.4.1 Communication

An exemplary implementation architecture for cellular network-based communication is represented in Figure E.2. A possible communication stack is shown in Figure E.3.

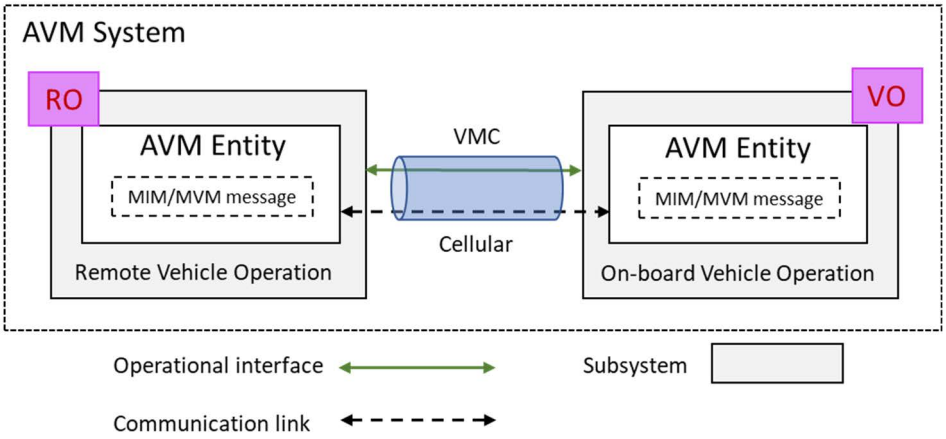


Figure E.2: Exemplary implementation architecture for cellular network-based communication

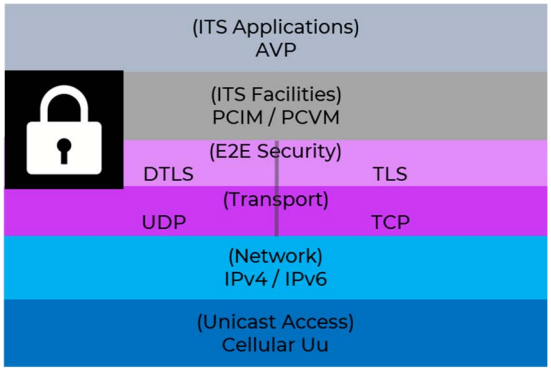


Figure E.3: IP protocol stack

The protocol stack in Figure E.3 is applicable when using cellular unicast communication between the RO and the VO subsystems.

E.4.2 Security

Depending on the transport layer protocol, i.e. Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), related protection and encryption are implemented for the ITS facilities layer using DTLS for UDP and TLS for TCP. Credentials for the secured communication is based on Recommendation ITU-T X.509 [i.17] certificates exchanged between trusted partners prior to the communication. An example sequence of communication and security establishment is shown in Figure E.4, the sequence also includes pre-requisites and steps executed prior to the execution of the actual driving to put this AVM facilities layer specification into a context.

The summarized steps are further described in 5GAA T-20002 [i.8].

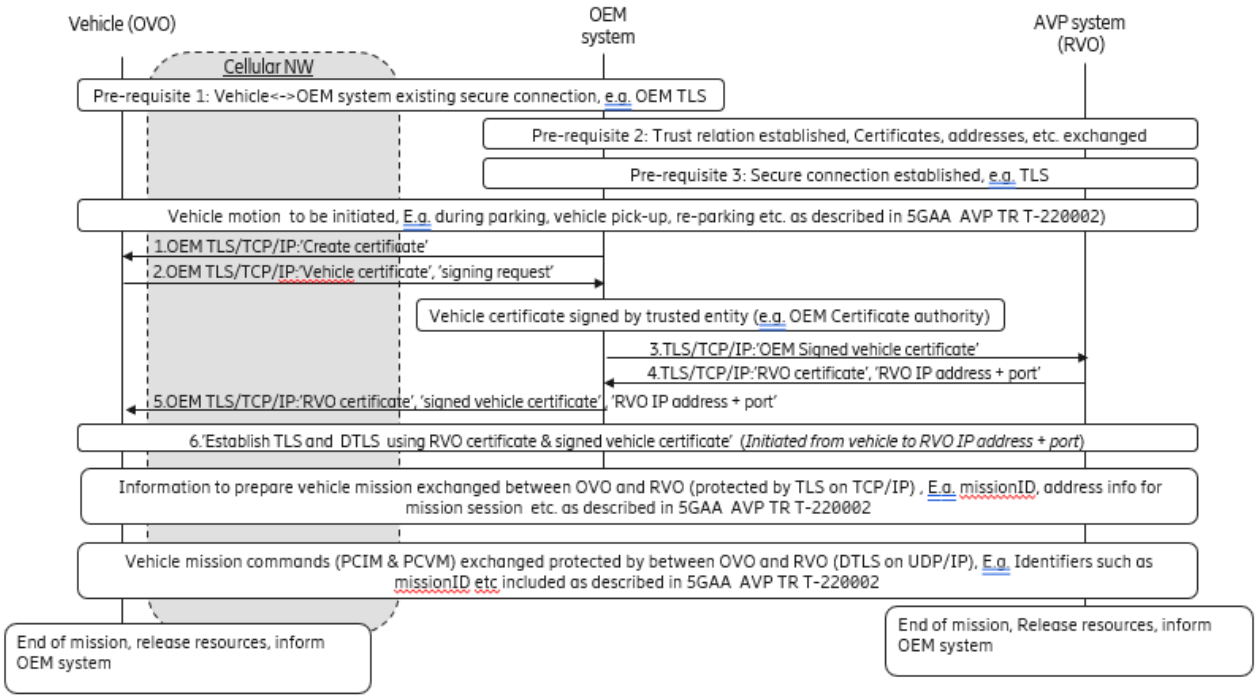


Figure E.4: Security and IP connectivity establishment

The illustrated sequence depicts a scenario where standard Information Technology (IT) security with encryption is applied on an E2E connection using IP as transport protocol, i.e. using X.509 certificates and mutual authentication between interacting actors.

The illustrated sequence show that certificates are exchanged during the AVM communication sequence, i.e. certificates with a short expiration time used that needs to be renewed for each mission (RO control session), the certificate handling could be done prior to the actual mission, which needs the certificate to have a longer validity time.

The session for the remote vehicle control is initiated from the vehicle, since the vehicle is the client (parking system is server) and knows when it is ready to allow remote control.

UDP (protected by DTLS) is used in this example for 'Vehicle mission commands' (MIM/MVM) to allow full application control for timers, retransmissions and message-based communication, etc. TCP could also be used. One or both sides may inform the OEM system about the completion of the mission and the end of the driving permission.

Above figure represents an example sequence and a partial signal flow for active driving of the vehicle in an AVM manoeuvre. The RO subsystem initializes the sequence by the *action* signal. The vehicle starts responding to action commands with an *operationMode* signal. Before the vehicle starts moving it signalizes its identification by a blinking code of the direction indicator lights. Actual driving starts when the RO subsystem sends a Path Snippet or a direct Control command. The nominal vehicle shutdown occurs when the destination is reached. Both the RO subsystem and the VO, subsystem can terminate the drive at any time.

NOTE: Figure E.4 depicts exemplary a non-detailed signal flow without timing conditions - only for reference.

E.5 C-ITS broadcast communication and security

E.5.1 Communication

An exemplary implementation architecture for cellular network-based communication is represented in Figure E.5 as described in 5GAA T-220002 [i.8].

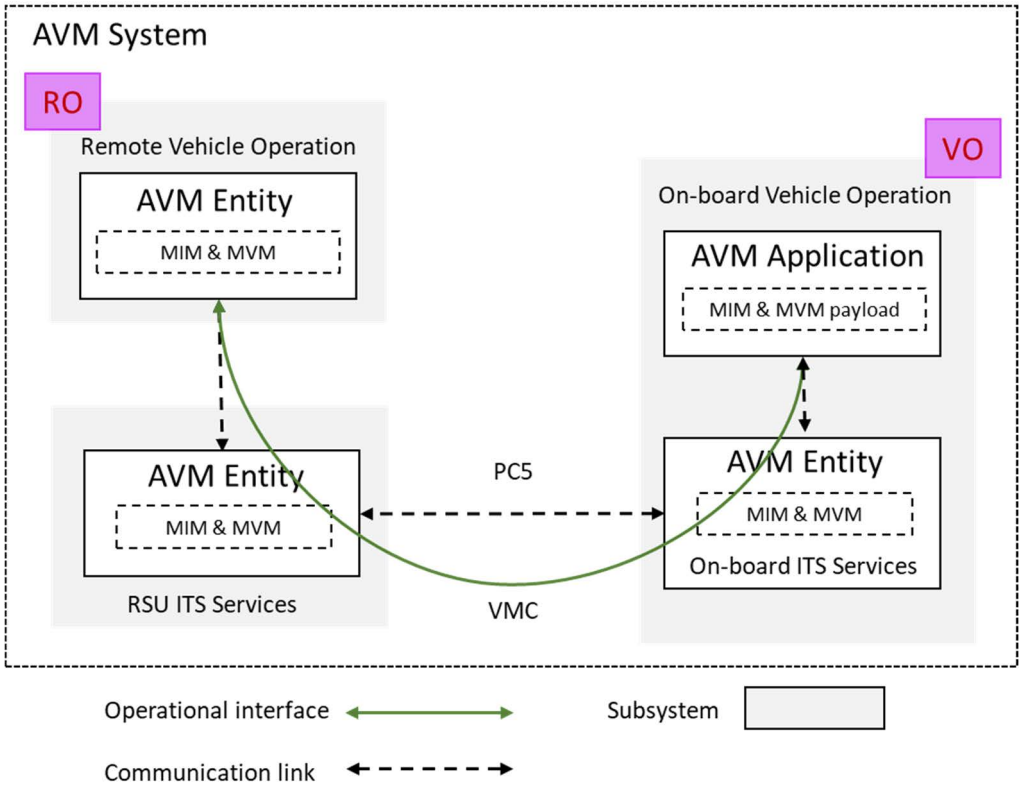


Figure E.5: Exemplary implementation architecture for PC5-based direct communication

The protocol stack in Figure E.6 is applicable when using e.g. PC5 broadcast communication between the RSU and the OBU.

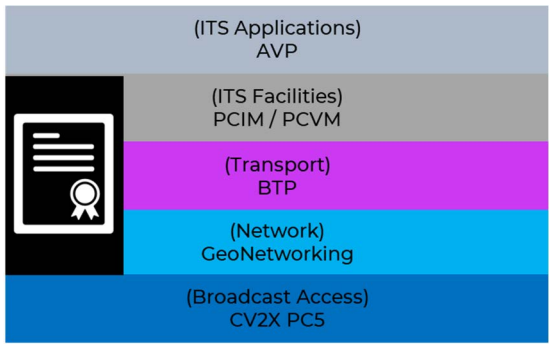


Figure E.6: Short range protocol stack

E.5.2 Security

References to ITS security are available ETSI TS 102 940 [i.13] and ETSI TS 103 097 [i.16].

Security services for standardised direct V2X communications are further-on defined in IEEE 1609.2 [i.5]. The main objectives of these services are:

- Authenticity - assurance that the sender is who they claim to be.
- Authorisation - assurance that the sender is entitled to the privileges they request.
- Integrity - assurance that any changes to the packet after it is signed can be detected.
- Anonymity - mitigating privacy risk of the vehicle user.

Authenticity, authorisation, and integrity are achieved by using a scheme based on digital signatures and certificates which are received from a trusted authority recognised by all network users.

However, use of fixed certificates to prove authorisation would allow an undetected third party to use the transmitted certificates to keep track of a particular vehicle; all that is needed is reception of the packets - the certificate and the data are available in plaintext. This would violate the need for privacy (as the tracker would know the whereabouts of any particular vehicle). As a result, vehicles do not receive or use permanent certificates, but instead use short-term pseudonym certificates that are changed every few minutes.

A high-level summary of the scheme is as follows:

- A valid V2X station (e.g. OBU, RSU) undergoes a registration process at the relevant Public Key Infrastructure (PKI) Certificate Authority (CA).
- As part of the above process, there is an exchange of public keys between the V2X station and the CA.
- The relevant CA provides a large number of pseudonym certificates to the V2X station, which are themselves signed by the CA; each certificate contains a different V2X station public key.
- During the operational stage, the V2X station signs the outgoing message using its private keys, and sends them along with the relevant certificate (or a hash of it).
- Receiving vehicles will validate the received certificate, and if valid, use it to then validate the digital signature for each received message. At this point the messages are cryptographically verified (there are further validation tests to the message, including time and location feasibility checks).
- At some point the V2X station will run out of short-term certificates and a renewed session with the PKI CA will be necessary to replenish them.

NOTE: If needed for compliance with privacy regulations, additional mechanisms can be added to help mitigate privacy issues.

EXAMPLE: Employing User Consent Agreement for AVM use.

E.6 PathSnippet Control - Implementation Considerations

E.6.1 Introduction

Clause E.6 describes the geometrical path interface. In the path interface, the RO provides a path and the current pose to the SV and the SV follows that path using its own controller.

The RO is designed to generate paths that enable seamless driving. It takes into account various vehicle limitations, including minimum and maximum curvature, maximum update frequency, and anticipated distances between waypoints. By considering these factors, the RO algorithm ensures that the generated paths are well-suited to the vehicle's capabilities, resulting in a smoother and more efficient driving experience.

The RO provides the SV with a new `DetectedVehiclePose` at an agreed interval, with a defined level of accuracy. However, it is important to note that the last known vehicle pose received from the RO can be several hundred milliseconds old, which can impact the accuracy of the control loop. To address this issue, the SV can use the timestamp in `DetectedVehiclePose` and its own odometry to extrapolate the driven distance since the RO captured the pose. By doing so, the SV can improve the accuracy of the data, resulting in more precise and reliable information.

If a new `PathSnippet` is not received early enough, the SV will come to a stop at the end of the current `PathSnippet` and wait for the RO to send a new one.

The velocity given in each waypoint is a maximum velocity. The SV reduces its velocity towards the end of the current `PathSnippet` appropriately in a way that it reaches 0 m/s at the last waypoint of the current `PathSnippet`, i.e. come to a full stop.

When an obstacle is detected on the desired path, the RO has two options to avoid a collision comfortably:

1) Compute a new path that bypasses the obstacle and transmit a new `PathSnippet` to the SV. The RO calculates the new path based on the current state of the SV and the surrounding environment, ensuring that the new path remains safe and feasible.

2) Shorten the current path so that the `PathSnippet` concludes with a safe distance from the obstacle. The RO reduces the length of the current path to ensure that the SV can come to a stop before reaching the obstacle.

E.6.2 Coordinate System

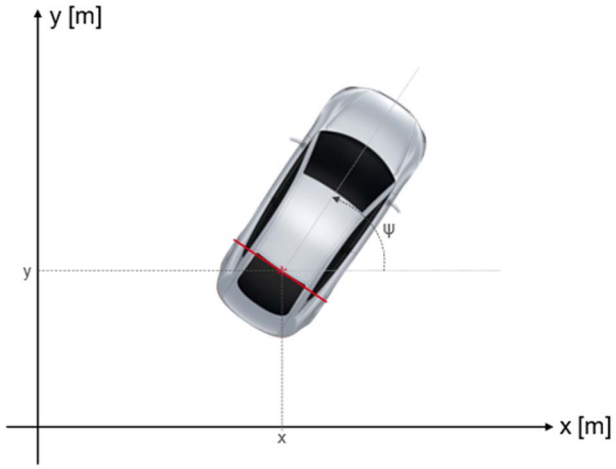


Figure E.7: Coordinate system

All coordinates are given in a fixed parking facility coordinate system as shown in Figure E.7. The parking facility coordinate system is a cartesian coordinate system with fixed origin. Coordinates can have both positive and negative values.

Poses always point to the centre of the rear axle of the SV. The orientation $\psi \in [0,2\pi)$ defines the orientation of the SV relative to the x-axis. Positive values define a rotation counter clockwise.

E.6.3 Curvature

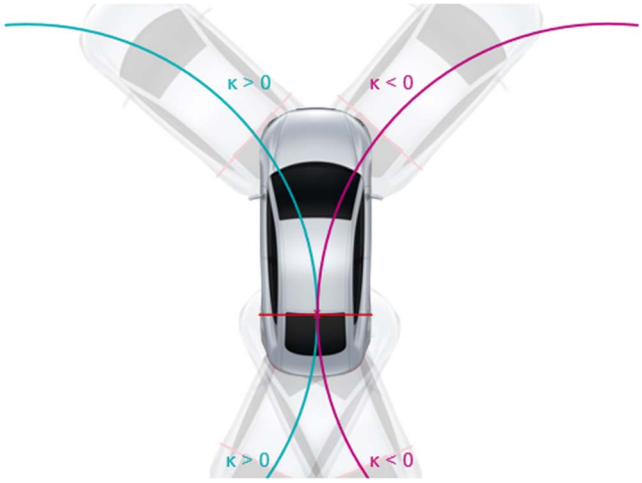


Figure E.8: Curvature description

Figure E.8 describes the curvature κ describes driving on a circle with radius $r=1/\kappa$:

- $\kappa = 0$ corresponds to driving straight.
- $\kappa > 0$ corresponds to driving left.
- $\kappa < 0$ corresponds to driving right.

For slow speeds as used by AVM, the curvature κ is related to the steering angle δ using wheelbase l as defined by the Ackermann steering formula.

The steering angle δ equals the front axis steering angle δF. The rear axis steering angle is fixed:

$$\delta R = 0$$

E.7 Trajectory Control - Implementation considerations

E.7.1 Introduction

As described by clause 7.7, the structure of the `trajectoryControl` container within the MIM message contains:

- `timeReference` data element
- (optional) `driveDirection` data element
- `ControlTrajectory` sequence
- (optional) `StateTrajectory` sequence

As illustrated by Figure 25 the *driveDirection* DE is optional. Nevertheless, to allow for vehicle motion control, using trajectory control interface, the direction of travel is mandatory. Therefore, in case of missing drive direction data element, the vehicle is served using another indicator included in received MIM.

The concept of trajectory leads to minimal requirements on the vehicle and relies on measurements and calculations based in the infrastructure of the parking spot.

The optional `StateTrajectory` sequence allows for error correction in actuator control, using the Cartesian coordinate system based on the reference point of the vehicle and on-board measurements. This is mended to be calculated as low-latency feedback control by the vehicle itself.

To allow the SV to control its actuators, the received control trajectory elements (e.g. curvature and acceleration) are evaluated to identify the appropriate control elements.

Alternatively, it is possible to use the DE *curvature*, velocity and optionally *DistanceToStop* for the vehicle motion control.

The appropriate data element within the control point sequence needs to be selected based on the time reverence for a given trajectory control sequence and the synchronized functional system clock. This functionality is described in clause E.7.2.

E.7.2 Control Point Selector

The Control Point Selector maps an execution time on each discrete `controlPoint` element, based on:

- `timeReference` element.
- `controlPoint` interval.
- Synchronized functional system clock.

Based on the *timeReference* element, stating the expected execution time of the first ControlTrajectory/`StateTrajectory` element, and the `controlPoint` interval (e.g. 40 ms), each `controlPoint` element is mapped on discrete point in time. From the resulting trajectory, the Control Point Selector provides, based on the current time, the appropriate `controlPoint` container element for actuator control.

If the current time falls between two discrete elements of the trajectory, the Control Point Selector interpolates the trajectory by using linear interpolation

NOTE: Trajectory interval is part of vehicle capabilities exchanged in service discovery and registration phase between AVMS and vehicle. For further description on capabilities exchange in registration and discovery phase, refer to 5GAA T-20002 [i.8] and for description of trajectory interval refer to [i.7].

History

Document history		
V2.1.1	May 2024	Publication