INTERNATIONAL STANDARD

ISO 13185-2

First edition
2015-04-01

Intelligent transport systems — Vehicle interface for provisioning and support of ITS services —

Part 2: Unified gateway protocol (UGP) requirements and specification for vehicle ITS station gateway (V-ITS-SG) interface

Systèmes intelligents de transport — Interface véhicule pour la fourniture et le support de services ITS —

Partie 2: Exigences de protocole et spécification pour l'interface passerelle de la station ITS du véhicule

Reference number
ISO 13185-2:2015(E)

Contents

Page

Foreword vi
Introduction vii
1 Scope 1
2 Normative references 1
3 Terms, definitions, symbols, and abbreviated terms 1
3.1 Terms and definitions 1
3.2 Abbreviated terms 2
4 Conventions 3
5 Document overview 4
6 UGP application layer services 6
6.1 General 6
6.2 Service primitives 6
6.3 Application layer service primitives — Confirmed service 7
6.4 Format description of service primitives 8
6.4.1 General definition 8
6.4.2 Service request and service indication primitives 8
6.4.3 Service response and service confirm primitives 9
6.4.4 Service request-confirm and service response-confirm primitives 10
6.5 UGP service call with global reply message handling 10
6.6 UGP service call with scheduled-based reply message handling 11
6.7 UGP service data unit specification 12
6.7.1 Mandatory parameters 12
7 UGP application layer protocol in ASN.1 document interface 13
7.1 General definition 13
7.2 Common message data and structure 13
7.3 ASN.1 model 14
8 Unified gateway protocol (UGP) clusters 15
8.1 Overview 15
8.2 UGP service clusters and associated services 15
9 UGP service cluster 1 — Global services 16
9.1 GlobalPositiveReply 16
9.2 GlobalNegativeReply 16
9.3 StopService 17
9.3.1 Service description 17
9.3.2 Message ‘StopServiceCall’ 17
9.3.3 Positive reply 17
9.3.4 Error handling 17
9.3.5 Example 17
9.4 Reset 18
9.4.1 Service description 18
9.4.2 Message ‘ResetCall’ 18
9.4.3 Positive reply 18
9.4.4 Error handling 18
9.4.5 Example 18
10 UGP service cluster 2 — Authentication 19
10.1 Authentication 19
10.1.1 Service description 19
10.1.2 Message ‘AuthenticationCall’ 19
10.1.3 Message ‘AuthenticationReply’ 19
10.1.4 Error handling 20

10.1.5 Example ... 20

11 UGP service cluster 3 — Supported data ... 21

11.1 GetSupportedData ... 21

11.1.1 Service description ... 21

11.1.2 Message ‘GetSupportedDataCall’ ... 21

11.1.3 Message ‘GetSupportedDataReply’ ... 22

11.1.4 Error handling ... 23

11.1.5 Example without ECU data ... 23

11.1.6 Example with ECU data ... 24

11.1.7 ECU filtered example ... 25

11.1.8 Negative access type and data parameter property filtered example ... 26

12 UGP service cluster 4 — Data parameter access ... 26

12.1 GetValue ... 26

12.1.1 Service description ... 26

12.1.2 Message ‘GetValueCall’ ... 26

12.1.3 Message ‘GetValueReply’ ... 27

12.1.4 Error handling ... 28

12.1.5 Single get data parameter list example ... 28

12.1.6 Asynchronous conditioned example ... 29

12.1.7 Monitor example ... 30

12.1.8 Structure example ‘vehicle info’ ... 31

12.2 SetValue ... 32

12.2.1 Service description ... 32

12.2.2 Message ‘SetValueCall’ ... 32

12.2.3 Positive reply ... 33

12.2.4 Error handling ... 33

12.2.5 Example ... 33

12.3 ControlValue ... 34

12.3.1 Service description ... 34

12.3.2 Message ‘ControlValueCall’ ... 34

12.3.3 Message ‘ControlValueReply’ ... 34

12.3.4 Error handling ... 35

12.3.5 Example 1 ... 35

12.3.6 Example 2 ... 38

13 UGP service cluster 5 — Diagnostic trouble code information access ... 39

13.1 GetDtcInfo ... 39

13.1.1 Service description ... 39

13.1.2 Message ‘GetDtcInfoCall’ ... 39

13.1.3 Message ‘GetDtcInfoReply’ ... 40

13.1.4 Error handling ... 41

13.1.5 Examples ... 41

13.2 ClearDtcInfo ... 42

13.2.1 Service description ... 42

13.2.2 Message ‘ClearDtcInfoCall’ ... 42

13.2.3 Positive reply ... 42

13.2.4 Error handling ... 42

13.2.5 Example ... 42

14 UGP service cluster 6 — In-vehicle network access ... 43

14.1 EnablePassThru ... 43

14.1.1 Service description ... 43

14.1.2 Message ‘EnablePassThruCall’ ... 43

14.1.3 Positive reply ... 44

14.1.4 Error handling ... 44

14.1.5 Example ... 44

15 UGP service cluster 7 — Maintenance ... 45

15.1 ListFile ... 45

15.1.1 Service description ... 45
15.1.2 Message 'ListFileCall' ... 45
15.1.3 Message 'ListFileReply' ... 46
15.1.4 Error handling ... 46
15.1.5 Example ... 46
15.2 ManageFile ... 47
15.2.1 Service description ... 47
15.2.2 Message 'ManageFileCall' ... 47
15.2.3 Message 'ManageFileReply' ... 48
15.2.4 Error handling ... 48
15.2.5 Example download ... 48
15.2.6 Example upload ... 49
15.2.7 Example delete ... 50

Annex A (normative) **Vehicle Interface Data Format definition (VIDF)** ... **51**

Annex B (normative) **Unified gateway protocol ASN.1 definition** ... **70**

Bibliography ... **78**

Foreword

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

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

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

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

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/TC 204, *Intelligent transport systems*.

ISO 13185 consists of the following parts, under the general title *Intelligent transport systems (ITS) — Vehicle interface for provisioning and support of ITS services*:

— *Part 1: General information and use case definition*

— *Part 2: Unified gateway protocol (UGP) requirements and specification for vehicle ITS station gateway (V-ITS-SG) interface*

The following parts are under preparation:

— *Part 3: Unified gateway protocol (UGP) server and client API specification*

— *Part 4: Unified gateway protocol (UGP) conformance test specification*

Introduction

This part of ISO 13185 has been established to define the requirements of a common software interface to a vehicle gateway to easily exchange vehicle information data amongst nomadic and/or mobile device, vehicle gateway, and the vehicle's Electronic Control Units (ECUs).

Applications supporting service provision use via nomadic and mobile devices need vehicle information data through an in-vehicle interface access method, as well as the harmonization of existing standards to support a single vehicle data access solution.

This document defines an ASN.1-based protocol between the nomadic and/or mobile device (ND) and the UGP Server (implemented in the V-ITS-SG) in the vehicle.

To achieve this, it is based on the Open Systems Interconnection (OSI) Basic Reference Model specified in ISO/IEC 7498-1 and ISO/IEC 10731, which structures communication systems into seven layers.

This part of ISO 13185 can be used by vehicle manufacturers for future vehicle design to support the design of vehicle gateways to interface with NDs.

The ND applications need vehicle information data through an in-vehicle interface access method (V-ITS-S with V-ITS-SG).

This part of ISO 13185 supports ITS applications which are based on ND in vehicles to operate on a common software interface to a V-ITS-SG to easily exchange vehicle information data among ND, vehicle V-ITS-SG, and ECUs.

The protocol implementation in the vehicle gateway features the following:

— the deny of access to the vehicle gateway data by unauthorized on-board and off-board test equipment;

— the deny of access to parts of the vehicle gateway data by unauthorized on-board and off-board test equipment (privacy);

— the identification of the vehicle gateway and the vehicle it is installed in;

— the list of in-vehicle connected ECUs to the vehicle gateway and their data parameters;

— methods to configure the access to vehicle data.

Intelligent transport systems — Vehicle interface for provisioning and support of ITS services —

Part 2: Unified gateway protocol (UGP) requirements and specification for vehicle ITS station gateway (V-ITS-SG) interface

1 Scope

This part of ISO 13185 specifies the requirements of an ASN.1-based protocol between a vehicle-ITS-Station Gateway (V-ITS-SG) and a nomadic and/or mobile device (ND) to easily exchange vehicle information data.

The ASN.1-based protocol has been specified to support a wired or wireless connection between the ND and V-ITS-SG.

The Unified Gateway Protocol (UGP) is used between the V-ITS-SG and the ND. UGP supports several features in order to provide

— authorization (data privacy),

— secured access,

— V-ITS-SG and in-vehicle ECUs identification,

— real-time vehicle data parameters with identifier and type information in ASN.1 format, and

— enhanced vehicle data parameters with identifier and type information in ASN.1 format.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 13185-1, *Intelligent transport systems — Vehicle interface for provisioning and support of ITS services — Part 1: General information and use case definition*

ISO 14229-2, *Road vehicles — Unified diagnostic services (UDS) — Part 2: Session layer services*

3 Terms, definitions, symbols, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 13185–1 and the following apply.

3.1.1
authentication
cryptographic service that provides assurance that the sender of a communication is who they claim to be

Note 1 to entry: Authentication might involve confirming the identity of a person or software program, tracing the origins of an artefact, ensuring that a product is what its packaging and labelling claims to be.

3.1.2
authorization
right granted by an authority

3.1.3
application layer protocol data unit
A_PDU
group of information added or removed by a layer of the OSI model

Note 1 to entry: Each layer in the model uses the PDU to communicate and exchange information, which can only be read by the peer layer on the receiving device and is then handed over to the next upper layer after stripping.

Note 2 to entry: The A_PDU (Application layer Protocol Data Unit) is directly constructed from the A_SDU (Application layer Service Data Unit) and the layer specific control information A_PCI (Application layer Protocol Control Information).

3.1.4
application layer service data unit
A_SDU
unit of data that has been passed down from an OSI layer to a lower layer and that has not yet been encapsulated into a protocol data unit (PDU) by the lower layer

Note 1 to entry: It is a set of data that is sent by a user of the services of a given layer and is transmitted semantically unchanged to a peer service user. A_SDU is the SDU in the OSI layer 7 (application layer).

3.1.5
privacy
choice made by the vehicle owner to grant information access (refer to directive 2002/58/EC dated 12 July 2002).

3.1.6
UGP Client
client implementing the UGP services

Note 1 to entry: For example, ND.

3.1.7
UGP Server
server implementing the UGP services

Note 1 to entry: For example, V-ITS-SG.

3.1.8
unified gateway protocol
UGP
application layer protocol to enable a *UGP Client* (3.1.6) to access data from the *UGP Server* (3.1.7)

3.2 Abbreviated terms

API	application programming interface
ASN.1	abstract syntax notation one
A_PDU	application layer protocol data unit

API	application programming interface
A_SDU	application layer service data unit
BT	bluetooth
C	conditional
C-ITS-S	central-intelligent transport system-station
CRC	cyclic redundancy check
Cvt	convention (M, O, C)
DTC	diagnostic trouble code
ECU	electronic control unit
ITS	intelligent transport systems
HCI	host controller interface
L2CAP	logical link control and adaptation protocol
LC	baseband link controller
LMP	link manager protocol
M	mandatory
ND	nomadic device
O	optional
OBEX	object exchange
PER	packed encoding rules
P-ITS-S	personal- intelligent transport system-station
RFCOMM	radio frequency communication (serial emulation API)
R-ITS-S	roadside- intelligent transport system-station
SDP	service discovery protocol
UDS	unified diagnostic services
UGP	unified gateway protocol
U-PER	unaligned packed encoding rules
VIN	vehicle identification number
V-ITS-SG	vehicle-intelligent transport system-station gateway

4 Conventions

This part of ISO 13185 is based on the conventions discussed in the OSI Service Conventions (ISO/IEC 10731:1994) as they apply for communication services.

These conventions specify the interactions between the service user and the service provider. Information is passed between the service user and the service provider by service primitives, which can convey parameters.

The distinction between service and protocol is summarized in Figure 1.

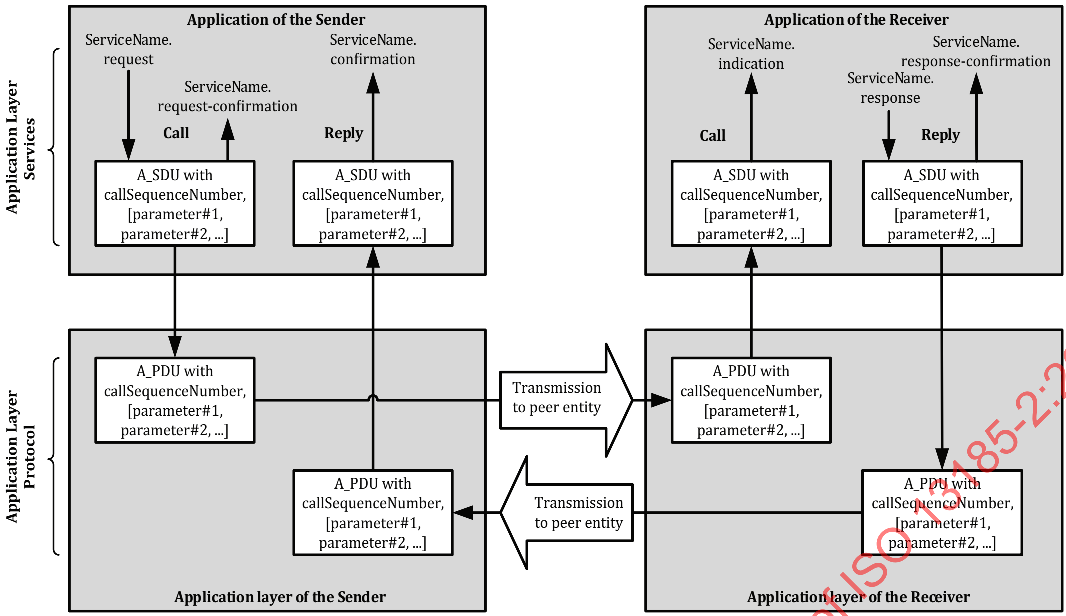


Figure 1 — Services and the protocol

This part of ISO 13185 defines services using the six service primitives: request, req_confirm, indication, response, rsp_confirm, and confirmation.

The request and indication service primitives always have the same format and parameters. Consequently for all services, the response and confirmation service primitives (except req_confirm and rsp_confirm) always have the same format and parameters. When the service primitives are defined in this part of ISO 13185, only the request and response service primitives are listed.

5 Document overview

Figure 2 shows the UGP document structure and automotive protocols and interfaces.

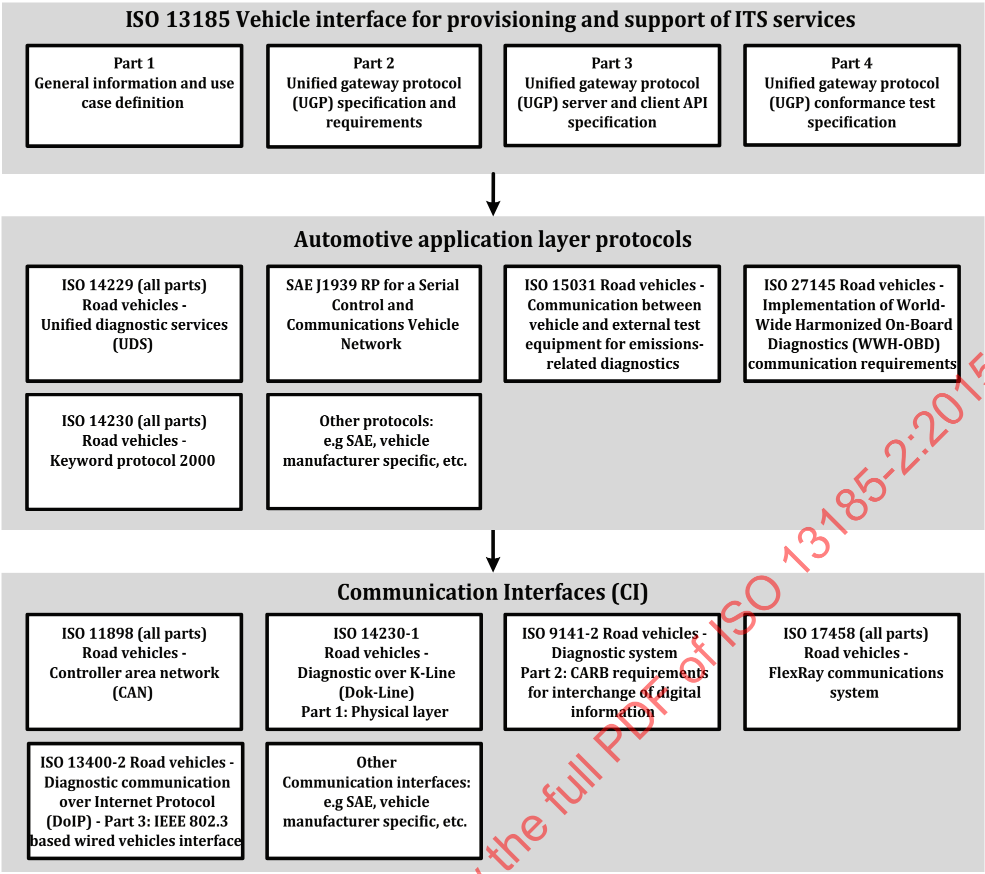


Figure 2 — UGP document structure and automotive protocols and interfaces

Figure 3 illustrates the implementation of the UGP and UDS document reference according to OSI model into a V-ITS-SG. Both UGP and UDS use the same session layer services as defined in ISO 14229-2. Non-UDS based protocols use their own session, transport, and network layer services.

Communication services as they apply to the OSI Service Conventions:

— Application layer:
This part of ISO 13185 specifies the requirements of an ASN.1 based UGP independent of underlying data link.

— Session layer:
ISO 14229-2 specifies the session layer services. This part of ISO 13185 uses the same session layer services as ISO 14229-1.

— Transport and Network layer:
Each communication link provides its own set of transport protocols and network layer services.

— Data link and Physical layer:
Each communication link provides its own set of data link and physical layer services.

The OSI layered concept implemented for the ISO 14229 series is the same as the one used for the ISO 13185 series.

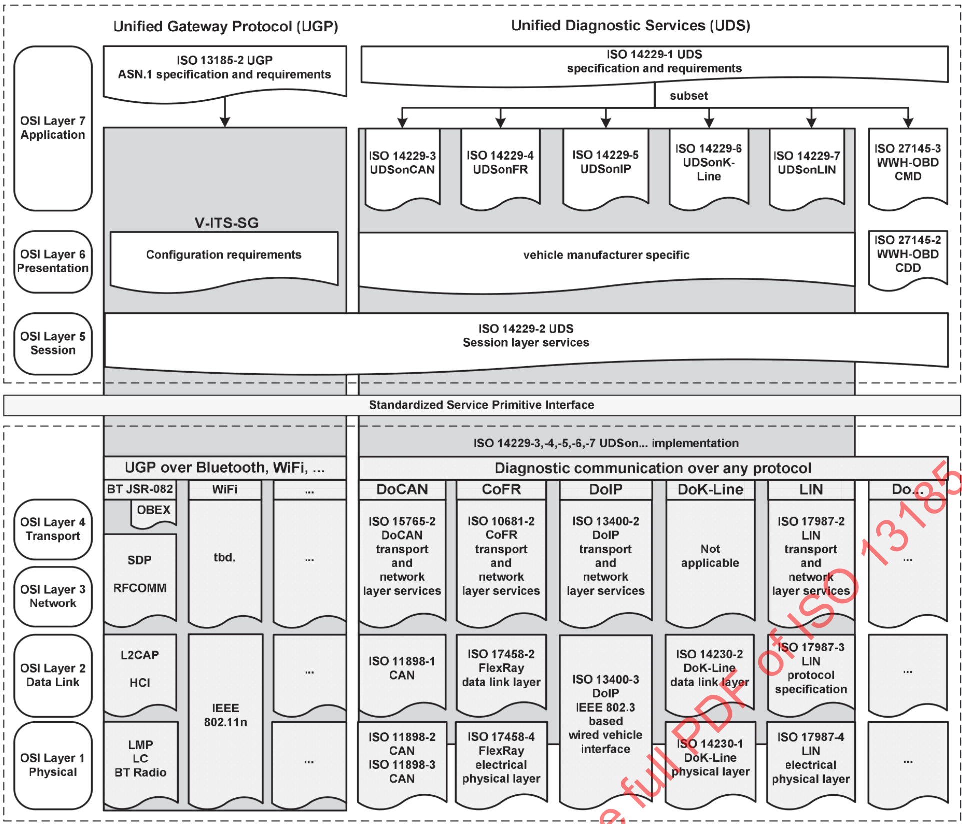


Figure 3 — Implementation of UGP document reference according to OSI model

6 UGP application layer services

6.1 General

Figure 4 illustrates the connection between the ND and V-ITS-SG. UGP provides application layer services to exchange the data between the Nomadic Device and the V-ITS-SG.

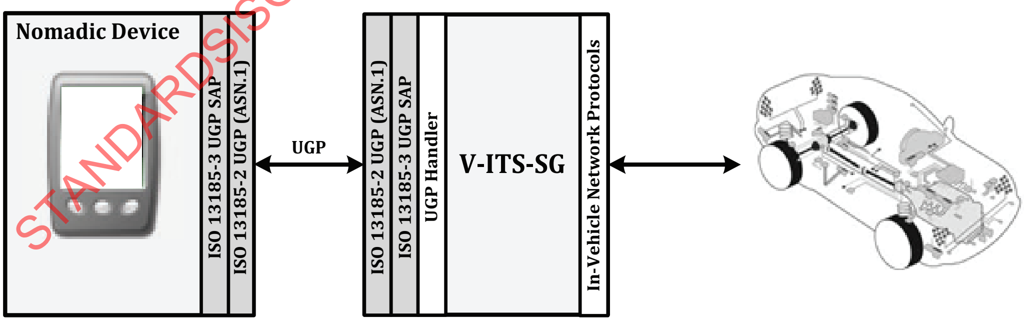


Figure 4 — Connection between ND and V-ITS-SG

6.2 Service primitives

Application layer services are referred to as Unified Gateway Protocol (UGP) services. The application layer services are used in client-server based systems to perform functions such as test, inspection, monitoring or diagnosis of on-board vehicle servers. The UGP Client, referred to as the P-ITS-S, R-ITS-S, or C-ITS-S, uses the application layer UGP services to request data from the UGP Server. The UGP Server uses the application layer services to send response data, provided by the requested UGP service, back to the UGP Client. The usage of application layer UGP services is independent from the UGP Client being an off-board or on-board tester. It is possible to have more than one UGP Client connected.

The UGP server and client API provides a number of services that all have the same general structure. For each service, six service primitives are specified:

— a **service request primitive**, used by the client function in the UGP application, to pass data about a requested UGP service to the diagnostics application layer;

— a **service request-confirmation primitive**, used by the client function in the UGP application, to indicate that the data passed in the service request primitive is successfully sent on the V-ITS-SG the nomadic device is connected to;

— a **service indication primitive**, used by the UGP application layer, to pass data to the server function of the server UGP application;

— a **service response primitive**, used by the server function in the V-ITS-SG UGP application, to pass response data provided by the requested UGP service to the UGP application layer;

— a **service response-confirmation primitive**, used by the server function in the V-ITS-SG UGP application, to indicate that the data passed in the service response primitive is successfully sent on the V-ITS-SG the nomadic device received the UGP request on;

— a **service confirmation primitive** used by the UGP application layer to pass data to the client function in the UGP application.

6.3 Application layer service primitives — Confirmed service

For a given service, the request-confirmation primitive and the response-confirmation primitive always have the same service data unit. The purpose of these service primitives is to indicate the completion of an earlier request or response service primitive invocation.

The response-confirm primitive is used by the application layer to indicate an internal event, which is significant to the server application, and pass communication results of an associated previous service response to the server function in the V-ITS-SG application.

Figure 5 depicts the application layer service primitives – Confirmed service.

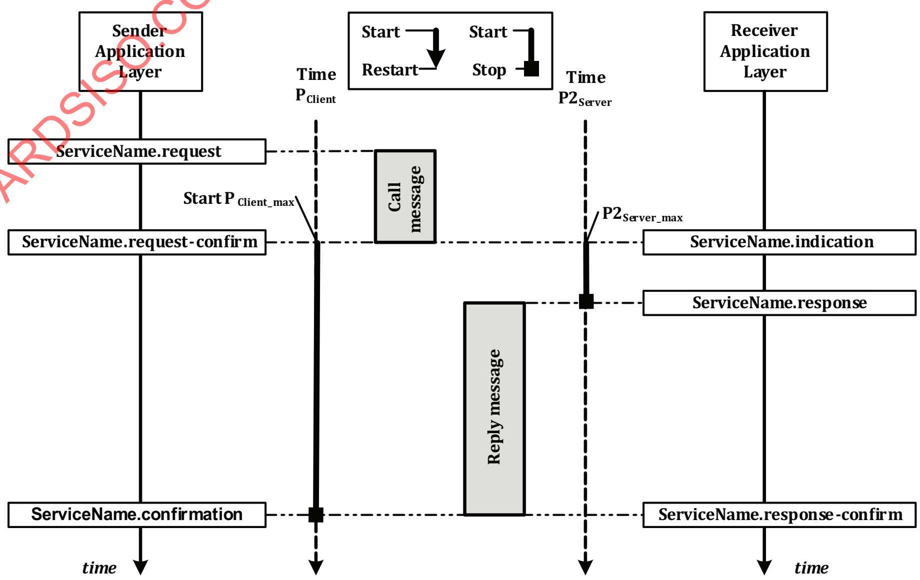


Figure 5 — Application layer service primitives – Confirmed service

6.4 Format description of service primitives

6.4.1 General definition

All application layer services have the same general format. Service primitives are written in the form:

```
service_name.type        (
                         parameter A, parameter B, parameter C
                         [,parameter 1, ...]
                         )
```

where

"service_name"	is the name of the UGP service (e.g. GetSupported-Data),
"type"	indicates the type of the service primitive (e.g. request),
"parameter A, ..."	is the A_SDU (Application layer Service Data Unit) as a list of values passed by the service primitive (addressing information),
"parameter A, parameter B, parameter C"	are mandatory parameters that shall be included in all service calls,
"[,parameter 1, ...]"	are parameters that depend on the specific service (e.g. parameter 1 can be the GetSupportedData service). The brackets indicate that this part of the parameter list can be empty.

6.4.2 Service request and service indication primitives

For each application layer service, service request and service indication primitives are specified according to the following general format:

```
service_name.request     (
                         callSequenceNumber,
                         timeInMillis,
                         length, data[, parameter 1,
                         ...],)
```

The request primitive is used by the client function in the application to initiate the service and pass data about the requested UGP service to the application layer.

```
service_name.indication  (
                         callSequenceNumber,
                         timeInMillis, length,
                         data[, parameter 1, ...],
                         )
```

The indication primitive is used by the application layer to indicate an internal event which is significant to the UGP Server application and pass data about the requested UGP service to the UGP Server function of the V-ITS-SG application.

The request and indication primitive of a specific application layer service always have the same parameters and parameter values. This means, that the values of individual parameters shall not be changed by the communicating peer protocol entities of the application layer when the data are transmitted from the UGP Client to the UGP Server. The same values that are passed by the client function in the UGP Client application to the application layer in the service request call shall be received by the UGP Server function of the application from the service indication of the peer application layer.

6.4.3 Service response and service confirm primitives

For each application layer service, service response and service confirm primitives are specified according to the following general format:

```
service_name.response        (
                             callSequenceNumber,
                             timeInMillis, length,
                             data[, parameter 1, ...],
                             )
```

The response primitive is used by the server function in the V-ITS-SG application to initiate the service and pass response data provided by the requested UGP service to the application layer.

```
service_name.confirm         (
                             callSequenceNumber,
                             timeInMillis,
                             length,data[, parameter 1,
                             ...],)
```

The confirm primitive is used by the application layer to indicate an internal event which is significant to the UGP Client application and pass results of an associated previous service request to the client function in the e.g. P-ITS-S application. It does not necessarily indicate any activity at the remote peer interface, e.g. if the requested service is not supported by the UGP Server or if the communication is broken.

The response and confirm primitive of a specific application layer service always have the same parameters and parameter values. This means that the values of individual parameters shall not be changed by the communicating peer protocol entities of the application layer when the data are transmitted from the UGP Server to the UGP Client. The same values that are passed by the server function of the V-ITS-SG application to the application layer in the service response call shall be received by the client function in the e.g. P-ITS-S application from the service confirmation of the peer application layer.

For each response and confirm primitive two different service data units (two sets of parameters) will be specified as the following:

— A positive response and positive confirm primitive shall be used with the first service data unit if the requested UGP service could be successfully performed by the server function in the V-ITS-SG;

— A negative response and confirm primitive shall be used with the second service data unit if the requested UGP service failed or could not be completed in time by the server function in the V-ITS-SG.

6.4.4 Service request-confirm and service response-confirm primitives

For each application layer service, service request-confirm and service response-confirm primitives are specified according to the following general format:

```
service_name.req_confirm          (
                                  callSequenceNumber,
                                  timeInMillis,
                                  result
                                  )
```

The request-confirm primitive is used by the application layer to indicate an internal event, which is significant to the UGP Client application, and pass communication results of an associated previous service request to the client function in the e.g. P-ITS-S application.

```
service_name.rsp_confirm          (
                                  callSequenceNumber,
                                  timeInMillis,
                                  result
                                  )
```

6.5 UGP service call with global reply message handling

The global reply message handling consists of the GlobalPositiveReply and GlobalNegativeReply messages. The UGP Server uses a GlobalNegativeReply message to inform a UGP client that the requested information is not accepted or supported. The GlobalPositiveReply is used by the UGP Server to positively respond to all UGP client calls without a call-specific reply message. These UGP services do not submit any data in the positive reply message.

Figure 6 shows the UGP service call with global reply message handling.

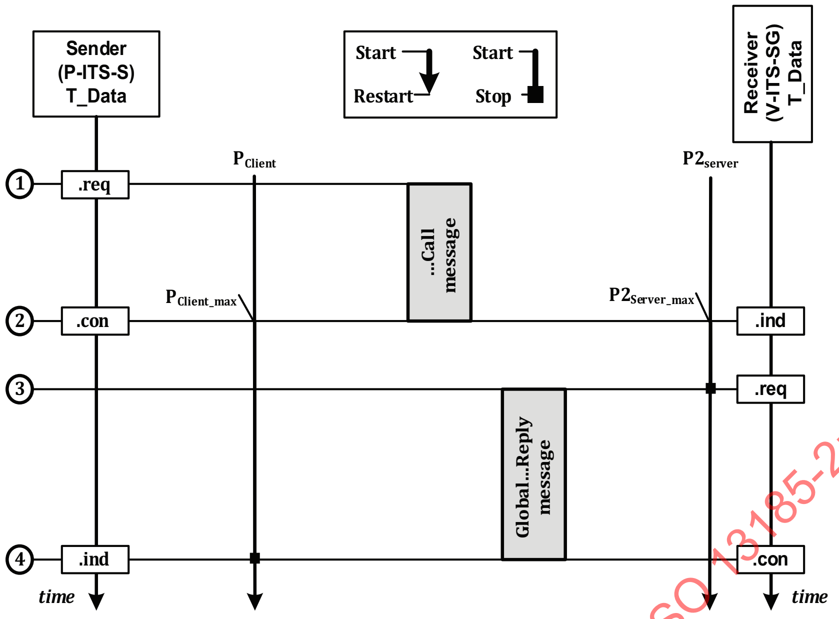
The following assumptions apply:

— The UGP Client is i.e. a P-ITS-S and the UGP Server is the V-ITS-SG.

— The T_Data interface for the UGP Client (P-ITS-S) and the UGP Server (V-ITS-SG) is specified in ISO 14229-2. It provides a data link independent interface to the session and application layer services.

— The `callMode` parameter is set to `normal`.

— The `refreshInterval` is set to the default value 0.



Key

1 **UGP Client (P-ITS-S) T_Data.req:** The i.e. P-ITS-S application of the UGP Client starts the transmission of a ...Call message by issuing a T_Data.req to its communication layer. The communication layer transmits the ...Call message to the UGP Server (V-ITS-SG).

2 **UGP Client (P-ITS-S) T_Data.con:** The completion of the ...Call message is indicated in the UGP Client (P-ITS-S) via T_Data.con. Now the response timing as described in ISO 14229-2 applies.

UGP Server (V-ITS-SG) T_Data.ind: The completion of the UGP Client (P-ITS-S) ...Call message is indicated in the UGP Server (V-ITS-SG) via the T_Data.ind. Now the response timing as described in ISO 14229-2 applies.

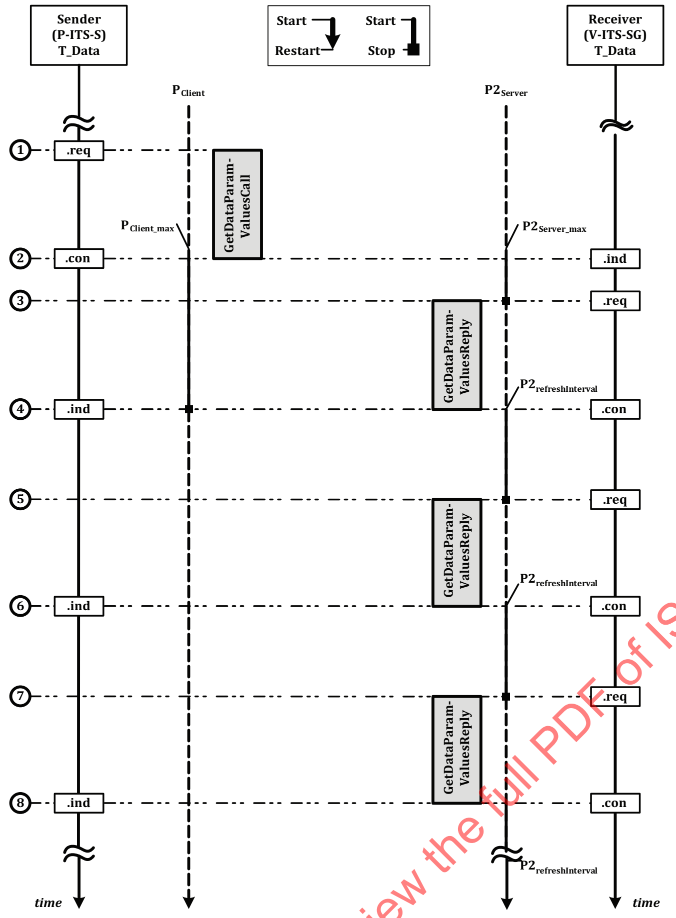
3 **UGP Server (V-ITS-SG) T_Data.req:** It is assumed that the UGP Client (P-ITS-S) requires a reply from the UGP Server (V-ITS-SG). The UGP Server (V-ITS-SG) shall transmit the ...Reply message to indicate that the ...Call has been processed and that the transmission of the reply has started.

4 **UGP Server (V-ITS-SG) T_Data.con:** The completion of the transmission of the ...Reply message is indicated in the UGP Server (V-ITS-SG) via T_Data.con.

Figure 6 — UGP service call with global reply message handling

6.6 UGP service call with scheduled-based reply message handling

Figure 7 shows the UGP service call with scheduled-based reply message handling.



Key

1 **UGP Client (P-ITS-S) T_Data.req**: application issues a `GetValueCall` message to the transport/network layer.

2 **UGP Server (V-ITS-SG) T_Data.ind**: transport/network layer issues to application the indication of the reception of a `GetValueCall` message.

UGP Client (P-ITS-S) T_Data.con: transport/network layer issues to application the confirmation of the completion of the `GetValueCall` message. UGP Client starts its P_{Client} timer using the default reload value $P_{Client} = P_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the communication media between UGP Server and UGP Client.

3/5/7 **UGP Server (V-ITS-SG) T_Data.req**: application has prepared the `GetValueReply` message and issues a T_Data.req to transport/network layer within $P2_{Server\_max}$. UGP Server stops the $P2_{Server}$ timer.

4/6/8 **UGP Server (V-ITS-SG) T_Data.con**: transport/network layer issues to application the completion of the `GetValueReply` message. UGP Server starts the $P2_{Server}$ timer using the value of $P2_{Server} = P2_{refreshInterval}$.

UGP Client (P-ITS-S) T_Data.ind: transport/network layer issues to application the reception of a `GetValueReply` message. UGP Client stops the P_{Client} timer.

Figure 7 — UGP service call with event- and scheduled-based reply message handling

6.7 UGP service data unit specification

6.7.1 Mandatory parameters

6.7.1.1 General definition

The application layer services contain three mandatory parameters. The following parameter definitions are applicable to all application layer services specified in this part of ISO 13185.

6.7.1.2 result

Type: enumeration

Range: ok, error

Description:

The parameter 'result' is used by the req_confirm and rsp_confirm primitives to indicate if a message has been transmitted correctly (ok) or whether the message transmission was not successful (error).

6.7.1.3 length

Type: 4 byte unsigned integer value

Range: $0_d - (2^{32}-1)_d$

Description:

This parameter includes the length of data to be transmitted/received.

6.7.1.4 data

This parameter includes all data to be exchanged by the higher layer entities.

6.7.1.5 callSequenceNumber

This parameter includes the sequence number of the call. The receiver shall use the same number for the reply.

6.7.1.6 timeInMillis

This parameter includes the time stamp of the reply data in milliseconds since 1970.

7 UGP application layer protocol in ASN.1 document interface

7.1 General definition

The communication between the UGP Client e.g. ND and the UGP Server (V-ITS-SG) is a unified gateway protocol with a high level of data abstraction. No knowledge about the vehicle protocols shall be required by the connected UGP Client.

The UGP Client application is pre-loaded with a vehicle-specific configuration in order to authenticate and interpret the data retrieved via the V-ITS-SG. The UGP Client displays or processes the requested information or relays the information to any kind of back office infra-structure, i.e. C-ITS-S. All information shall be composed of the V-ITS-SG and delivered to the UGP Client upon request if authentication and authorization criteria have been met by the UGP Client.

7.2 Common message data and structure

A UGP service consists of a request and response message. The data transmitted in the request and response message are structured in ASN.1. The data are encoded in Unaligned Packed Encoding Rules (U-PER). To distinguish the messages, all request messages end with 'Call' and all response messages end with 'Reply'.

Table 1 — Definition of UGPMessage

Msg	**UGPMessage**		
Attributes	**Name**	**Description**	**Cvt**
	`callSequenceNumber`	This parameter includes the sequence number of the call. The receiver shall use the same number for the reply.	M
	`timeInMillis`	This parameter includes the time stamp of the reply data in milliseconds since 1970.	O

ASN.1	`UGPMessage::= SEQUENCE {`
	`callSequenceNumber UNUM16,`
	`timeInMillis UNUM64 OPTIONAL,`
	`choiceUGP CHOICE {`
	`authenticationCall AuthenticationCall,`
	`authenticationReply AuthenticationReply,`
	`getSupportedDataCall GetSupportedDataCall,`
	`...`
	`globalPositiveReply GlobalPositiveReply,`
	`globalNegativeReply GlobalNegativeReply,`
	`...`
	`},`
	`...`
	`}`
	`ugpVersion Version::= 1`		

All ASN.1 messages have a common basic structure. The root element is called `UGPMessage`. The message and its attributes and the ASN.1 representations are defined in Table 1.

All services that have no positive response attributes use the global reply message `GlobalPositiveReply` (see 9.1) instead of an own reply message.

All services that cannot support, interpret, etc. the attributes included in the message call (request) shall send the negative response message `GlobalNegativeReply` (see 9.2).

7.3 ASN.1 model

The ASN.1 model consists of two modules:

— `VehicleInterfaceDataFormat.asn`: defines simple types like `Boolean`, `String`, `UNUM16` and `SNUM32`. Additionally `SEQUENCES` for the data parameter definition are introduced: `DisplayName`, `DataType`, `Numeric`, `EnumStringItem`, `DataParam`, `DataParamValue`, etc. (see B.1);

— `UnifiedGatewayProtocol.asn`: contains the `UGPMessage` with all of its `choiceUGP` like `authenticationCall`, `getSupportedDataCall` or `getDtcInfoReply` and the `VIDFConfig` (see B.2).

The ASN.1 model shall be used for validating the exchanged documents for a better understanding of the allowed structure.

8 Unified gateway protocol (UGP) clusters

8.1 Overview

Table 2 provides an overview of the UGP clusters. A UGP cluster can have one or more UGP services.

Table 2 — UGP clusters and associated services

# — Main title of cluster	Brief description	Related services	Cvt
1 — Global services	This cluster handles the positive response for all UGP services without response parameters and all negative responses. Asynchronous services can be stopped and the key can be resetted.	9.1 GlobalPositiveReply	—
		9.2 GlobalNegativeReply	—
		9.3 StopService	M
		9.4 Reset	M
2 — Authentication	The UGP services belonging to this cluster initiate the communication to the V-ITS-SG while checking the authentication and authorization.	10.1 Authentication	O[a]
3 — Supported data	The single UGP service belonging to this cluster requests the supported data parameters. This UGP service should be called once after authentication.	11.1 GetSupportedData	M[b]
4 — Data parameter access	The UGP services belonging to this cluster get, set and control the data parameter values in the V-ITS-SG (UGP Server). Getting and controlling can be scheduled in real time intervals by the V-ITS-SG.	12.1 GetValue	M
		12.2 SetValue	C[c]
		12.3 ControlValue	C[c]
5 — Diagnostic trouble code information access	The UGP services belonging to this cluster handle the reading and clearing of the diagnostic trouble codes and associated information.	13.1 GetDtcInfo	M
		13.2 ClearDtcInfo	M
6 — In-vehicle network access	The UGP services belonging to this cluster enable a switch to another protocol in the V-ITS-SG (UGP Server), which the vehicle manufacturer might choose to perform non-public configurations e.g. required during the assembly of the vehicle.	14.1 EnablePassThru	C[c]
7 — Maintenance	The UGP services belonging to this cluster maintain the V-ITS-SG (server) core software and its configuration with services to list, download, upload and delete. In addition, log files and snapshots can be listed and downloaded from the V-ITS-SG.	15.1 ListFile	C[c]
		15.2 ManageFile	C[c]

[a] The Authentication service is optional. If the service is not executed, the default access rights are active (see 10.1).

[b] The GetSupportedData service shall only report the default data parameters which are required by local legislation and appropriate for the vehicle type and model, if the Authentication service has not been executed.

[c] This service is conditional and depends on the user access rights of the AuthenticationReply. If the Authentication service has not been executed, a GlobalNegativeReply is returned from the V-ITS-SG.

8.2 UGP service clusters and associated services

Figure 8 shows the UGP service clusters and associated services. The services shown in cluster “authentication and encryption” and “identification and supported data” shall be executed in the sequence as shown prior to any services of the other clusters.

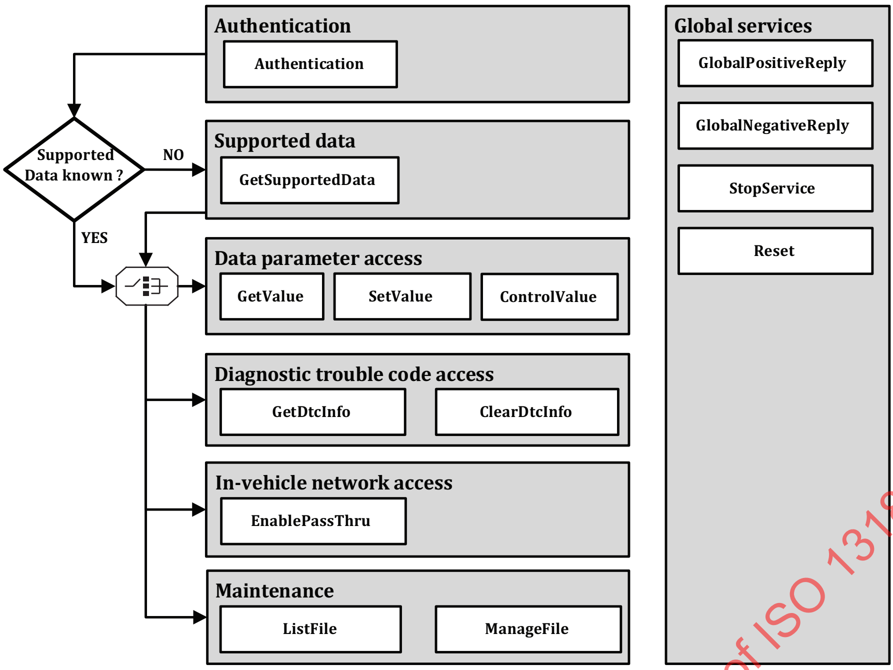


Figure 8 — UGP service clusters and associated services

9 UGP service cluster 1 — Global services

9.1 GlobalPositiveReply

Table 3 defines the global positive response message `GlobalPositiveReply`. List attributes are marked with {} in the attributes part of the message definition.

Table 3 — Definition of the message 'GlobalPositiveReply'

Msg	GlobalPositiveReply	Replies services requests without own reply message.	
Attributes	Name	Description	Cvt
	— no attributes —		

ASN.1	`GlobalPositiveReply::= SEQUENCE {`
	`...`
	`}`		

9.2 GlobalNegativeReply

Table 4 defines the negative response message `GlobalNegativeReply` for all services. It contains a list of `VIErrorValue` elements. A `VIErrorValue` consists of the mandatory attribute `errorId` as reference to an error. The number and content of the attributes `rvId` depend on the error. Examples for a `GlobalNegativeReply` are defined in the respective service.

Table 4 — Definition of the message 'GlobalNegativeReply'

Msg	GlobalNegativeReply	Possible reply on all ...Calls without results in the reply.	
Attributes	Name	Description	Cvt
	`error {}`	List of error values containing following sub attributes:	M
	— `errorId`	— Unique error identifier, references to an error	M

Table 4 *(continued)*

	— `attribute`	— List of data parameter values (see A.11)	O

ASN.1	`GlobalNegativeReply::= SEQUENCE {`
	`    error                SEQUENCE OF VIErrorValue,`
	`    ...`
	`}`		

9.3 StopService

9.3.1 Service description

The `StopService` is used to request the termination of a previously defined asynchronous service.

9.3.2 Message 'StopServiceCall'

Table 5 defines the request message `StopServiceCall` using the `callSequenceNumber` of a previously defined asynchronous service, i.e. a service requested by a call with attribute `refreshInterval` greater than 0. There is no relation between the `callSequenceNumber` of the `StopServiceCall` message and the attribute.

Table 5 — Definition of the message 'StopServiceCall'

Msg	**StopServiceCall**	Request a service termination.	
Attrib.	**Name**	**Description**	**Cvt**
	`callSequenceNumber`	The sequence number of the asynchrounous service to stop.	M

ASN.1	`StopServiceCall::= SEQUENCE {`
	`    callSequenceNumber  UNUM16,`
	`    ...`
	`}`		

9.3.3 Positive reply

If the requested service can be stopped, a `GlobalPositiveReply` (see 9.1) is sent.

9.3.4 Error handling

If the `StopService` fails with an invalid `callSequenceNumber` or a not stoppable service, a `GlobalNegativeReply` (see 9.2) is returned.

9.3.5 Example

An example for the `StopService` is included in Table 6. The UGP Client requests the termination of a service with `stopServiceCall` by defining the `callSequenceNumber 192`. The V-ITS-SG replies with a positive `stopServiceReply` with no additional information.

Table 6 — Example for service StopService

ASN.1	`stopServiceCall UGPMessage::= { callSequenceNumber 496,`
	`    choiceUGP stopServiceCall: { callSequenceNumber 192`
	`} }`
	`stopServiceReply UGPMessage::= { callSequenceNumber 496,`
	`    choiceUGP globalPositiveReply: {`
	`} }`

9.4 Reset

9.4.1 Service description

This `Reset` service is used to reboot or to shut down the V-ITS-SG.

9.4.2 Message 'ResetCall'

Table 7 defines the request message `ResetCall`.

Table 7 — Definition of the message 'ResetCall'

Msg	**ResetCall**	Request a key off and on reset.	
Attributes	**Name**	**Description**	**Cvt**
	`resetType`	Type of the reset: `reboot`, `shutdown`	M

ASN.1	`ResetCall::= SEQUENCE {`
	`    ...`
	`}`		

9.4.3 Positive reply

A positive reply is handled by a `GlobalPositiveReply` (see 9.1).

9.4.4 Error handling

If the key off and on reset does not work, a `GlobalNegativeReply` (see 9.2) is returned.

9.4.5 Example

An example for the `CodeReference` is included in Table 8. The UGP Client requests the reset with `resetCall`. The V-ITS-SG replies with a positive `resetReply` with no additional information.

Table 8 — Example for service Reset

ASN.1	`resetCall UGPMessage::= { callSequenceNumber 480,`
	`    choiceUGP resetCall: { resetType reboot`
	`} }`
	`resetReply UGPMessage::= { callSequenceNumber 480,`
	`    choiceUGP globalPositiveReply: {`
	`} }`

10 UGP service cluster 2 — Authentication

10.1 Authentication

10.1.1 Service description

The Authentication service starts the communication between the UGP Client (P-ITS-S / R-ITS-S) and the V-ITS-SG. It authorizes the client to the V-ITS-SG and supplies authorization access.

10.1.2 Message 'AuthenticationCall'

Table 9 defines the request message `AuthenticationCall`.

Table 9 — Definition of the message 'AuthenticationCall'

Msg	**AuthenticationCall**	Initiates the communication to the V-ITS-SG by requesting the users V-ITS-SG authentication key.	
Attributes	**Name**	**Description**	**Cvt**
	`authenticationKey`	Public key for access to a V-ITS-SG	M

ASN.1	`AuthenticationCall::= SEQUENCE {`
	`    authenticationKey   String,`
	`    ...`
	`}`		

10.1.3 Message 'AuthenticationReply'

Table 10 defines the response message `AuthenticationReply`. It is used to confirm the V-ITS-SG authentication key submitted by the UGP Client and to transfer all authorizations enabled by the key.

Table 10 — Definition of the message 'AuthenticationReply'

Msg	**AuthenticationReply**		Replies the `AuthenticationCall` by returning a bit mask for the users authorization.			
Attributes	**Name**		**Description**			**Cvt**
	`authorization`		Bit mask with all authorizations; uses the bit definition of the authorization bits:			M
	Bit	**Name**	**= 0 …**	**= 1 …**	**… Description**	**Default**
	0	get-value-extended-access	only default	all supported	service GetValue (see 12.1)	0
	1	set-value-access	no access	access	service SetValue (see 12.2)	0
	2	control-value-access	no access	access	service ControlValue (see 12.3)	0
	3	enable-pass-thru-access	no access	access	service EnablePassThru (see 14.1)	0
	4	file-download-access	no access	access	service ListFile (see 15.1), service ManageFile (see 15.2) with activityType = download	0
	5	file-upload-access	no access	access	service ManageFile (see 15.2) with activityType =upload	0
	6	file-delete-access	no access	access	service ManageFile (see 15.2) with activityType =delete	0

ASN.1

```
AuthenticationReply::= SEQUENCE {
     authorization         AuthorizationBits,
     ...
}
AuthorizationBits::= BIT STRING {
     get-value-extended-access   (0),
     set-value-access            (1),
     control-value-access        (2),
     enable-pass-thru-access     (3),
     file-download-access        (4),
     file-upload-access          (5),
     file-delete-access          (6)
} (SIZE(7, ...))
```


10.1.4 Error handling

If the authentication fails, a `GlobalNegativeReply` (see 9.2) is returned.

10.1.5 Example

A typical example for the Authentication is included in Table 11. The UGP Client (P-ITS-S/R-ITS-S) requests the Authentication (`authenticationCall`) with his V-ITS-SG authorization key. The V-ITS-

SG replies with a positive message (`authenticationPosReply`) including the bit mask for the authorization of the UGP Client.

If the UGP Client cannot be authenticated or is not authorized, a `GlobalNegativeReply` (see 9.2) is returned (see `authenticationNegReply`).

Table 11 additionally shows the unaligned PER encoding (U-PER) for the example in the lower area. The comments with leading '−−' are only added for readability.

Table 11 — Example for service Authentication

ASN.1	`authenticationCall UGPMessage::= { callSequenceNumber 1,`
	`    choiceUGP authenticationCall: {`
	`        authenticationKey "0vrmYdc6ziscBdMhGphiT6m0a0D2dNLCnETRSlg"`
	`} }`
	`authenticationPosReply UGPMessage::= { callSequenceNumber 1,`
	`    choiceUGP authenticationReply: {`
	`        authorization '1000100'B`
	`} }`
	`authenticationNegReply UGPMessage::= { callSequenceNumber 1,`
	`    choiceUGP globalNegativeReply: { error {`
	`        { errorId 1 }`
	`} } }`

Unaligned PER	`--authenticationCall (39 byte):`
	`00 00 40 13  B0 ED CB 6D  9C 98 DB 7A  D3 CF 1C 2C  93 74 47 E1`
	`A3 4D 46 DB  58 61 61 11  96 49 D3 21  EE 8B 52 95  3D 99 C0`
	
	`--authenticationPosReply (5 byte):`
	`00 00 41 3F  80`
	
	`--authenticationNegReply (9 byte):`
	`00 00 54 00  90 00 00 00  20`

11 UGP service cluster 3 — Supported data

11.1 GetSupportedData

11.1.1 Service description

This service is used to request the supported data parameters.

11.1.2 Message 'GetSupportedDataCall'

Table 12 defines the request message `'GetSupportedDataCall'`. All attributes are optional and are used as a filter of the expected response. If no attribute is defined, all supported data parameters of all supported ECUs are requested. The `supportedDataFilter vehicle-info-only` requests only vehicle info data parameters. The `supportedDataFilter with-ecu-data` requests all data parameters including their ecuId. The `supportedDataFilter without-ecu-data` requests all supported data parameters without ecuId. If the attribute `eculist` is added, the data parameters are filtered to the specified ECUs. If the `accessType` is defined, only data parameters of this access type are requested. If the `dataParamProperty` is defined, only data parameters of this data parameter property (e.g. sensor) are requested.

Table 12 — Definition of the message 'GetSupportedDataCall'

Msg	**GetSupportedDataCall**	Request the supported data parameters from the V-ITS-SG.	
Attributes	**Name**	**Description**	**Cvt**
	supportedDataFilter	Filters the supported data to: vehicle-info-only, with-ecu-data, without-ecu-data	M
	ecuList {}	List of the ECU identifer of the in-vehicle network ECUs from where the data parameters should be retrieved.	O
	accessType	Filters the supported data parameters to the specified access type (see A.7). If no accessType is defined, the access types are not filtered.	O
	dataParamProperty	Filters the supported data parameters to the specified data parameter property (see A.7). If no dataParamProperty is defined, the data parameter properties are not filtered.	O

ASN.1	GetSupportedDataCall::= SEQUENCE {
	supportedDataFilter SupportedDataFilter DEFAULT with-ecu-data,
	ecuList SEQUENCE OF Identifier OPTIONAL,
	accessType AccessType OPTIONAL,
	dataParamProperty DataParamProperty OPTIONAL,
	...
	}
	SupportedDataFilter::= ENUMERATED { vehicle-info-only, with-ecu-data,
	without-ecu-data, ...
	}		

11.1.3 Message 'GetSupportedDataReply'

Table 13 defines the positive response message `GetSupportedDataReply` which contains only the vehicle info data parameter `rvIds` if the `supportedDataFilter vehicle-info-only` is used. The filter `with-ecu-data` returns a list of data parameter mappings. The filter `without-ecu-data` returns only a list of the registered value identifiers (`rvIds`).

The definition of all possible data parameters (including data type, unit, etc.) is included in the `VIDFConfig` (see A.19). The `GetSupportedDataReply` returns the supported `rvIds` that can be filtered from the `VIDFConfig`.

Table 13 — Definition of the message 'GetSupportedDataReply'

Msg	GetSupportedDataReply	Replies the GetSupportedDataCall by returning the data types, data parameters, and their mappings to the ECUs.	
Attributes	**Name**	**Description**	**Cvt**
	dataParamList {}	List of all data parameter rvIds, if supportedDataFilter without-ecu-data; List of vehicle info data parameter rvIds, if supportedDataFilter vehicle-info-only	O1[a]
	dataParamMapping {}	List of mappings between data parameter and ECU (see A.8), if supportedDataFilter with-ecu-data.	O2[a]

ASN.1	GetSupportedDataReply::= SEQUENCE {
	dataParamList SEQUENCE OF Identifier OPTIONAL,
	dataParamMapping SEQUENCE OF DataParamMapping OPTIONAL,
	...
	}		

[a] Either O1 or O2 must be defined depending on the supportedDataFilter set in GetSupportedDataCall

In dataParamMapping, data parameters are mapped to ECUs to define 'this ECU does support this data parameter'.

11.1.4 Error handling

If no supported data parameters can be retrieved, a GlobalNegativeReply (see 9.2) is returned. If at least one data parameter is available but others cannot be retrieved, the GetSupportedDataReply contains a list of VIErrorValues.

11.1.5 Example without ECU data

Table 14 shows a positive example for the use of the service GetSupportedData without ECU data. The UGP Client requests all supported data parameters with the getSupportedDataCallWithoutEcuData using the supportedDataFilter without-ecu-data. If all supported data parameters can be retrieved, the UGP Server responds with a getSupportedDataReplyWithoutEcuData containing a list of all supported data parameter rvIds.

Table 14 — GetSupportedData example without ECU data

ASN.1	`getSupportedDataCallWithoutEcuData UGPMessage::= {`
	`    callSequenceNumber 112,`
	`    choiceUGP getSupportedDataCall: { supportedDataFilter without-ecu-data`
	`} }`
	
	`getSupportedDataReplyWithoutEcuData UGPMessage::= {`
	`    callSequenceNumber 112,`
	`    choiceUGP getSupportedDataReply: { dataParamList {`
	`        1002, 2341, 461, 10020, 10021, 10030, 10042, 10050`
	`} } }`

U-PER	`-getSupportedDataCallWithoutEcuData (4 byte):`
	`00 1C 02 04`
	
	`-getSupportedDataReplyWithoutEcuData (37 byte):`
	`00 1C 03 41  10 00 00 7D  50 00 01 24  B0 00 00 39  B0 00 04 E4  90 00 04 E4`
	`B0 00 04 E5  D0 00 04 E7  50 00 04 E8  40`

11.1.6 Example with ECU data

Table 15 shows a positive example for the use of the service `GetSupportedData` including ECU data. The UGP Client requests all supported data parameters with the `getSupportedDataCallWithEcuData` using the `supportedDataFilter  with-ecu-data`. If all supported data parameters can be retrieved, the UGP Server responds with a `getSupportedDataReplyWithEcuData` containing a list of all supported data parameter mappings.

Table 15 — GetSupportedData example with ECU data

ASN.1	getSupportedDataCallWithEcuData UGPMessage::= { callSequenceNumber 113,
	    choiceUGP getSupportedDataCall: { supportedDataFilter with-ecu-data
	} }
	getSupportedDataReplyWithEcuData UGPMessage::= { callSequenceNumber 113,
	    choiceUGP getSupportedDataReply: { dataParamMapping {
	        { rvId 1002, ecuId 17 }, { rvId 2341, ecuId 51 },
	        { rvId 10020, ecuId 21 }, { rvId 10021, ecuId 21 },
	        { rvId 10022, ecuId 21 }, { rvId 10023, ecuId 21 },
	        { rvId 10024, ecuId 21 }, { rvId 10042, ecuId 51 },
	        { rvId 10050, ecuId 51 }
	} } }

U-PER	-getSupportedDataCallWithEcuData (4 byte): 00 1C 42 02 -getSupportedDataReplyWithEcuData (78 byte): 00 1C 43 21 28 00 00 3E A8 00 00 01 14 00 00 49 2C 00 00 01 9A 00 00 9C 92 00 00 00 55 00 00 4E 4B 00 00 00 2A 80 00 27 26 80 00 00 15 40 00 13 93 C0 00 00 0A A0 00 09 CA 20 00 00 05 50 00 04 E7 50 00 00 06 68 00 02 74 28 00 00 03 30

11.1.7 ECU filtered example

Table 16 shows an ECU filtered example for the use of the service `GetSupportedData`. The UGP Client requests all supported data parameters for ECU 17 with the `getSupportedDataCallEcuFiltered`. If data parameters of ECU 17 can be retrieved, the UGP Server responds with a `getSupportedDataReplyEcuFiltered` containing a list of the data parameters of ECU 17.

Table 16 — ECU filtered example of GetSupportedData

ASN.1	getSupportedDataCallEcuFiltered UGPMessage::= { callSequenceNumber 114,
	    choiceUGP getSupportedDataCall: { supportedDataFilter with-ecu-data,
	    ecuList { 17 }
	} }
	getSupportedDataReplyEcuFiltered UGPMessage::= { callSequenceNumber 114,
	    choiceUGP getSupportedDataReply: { dataParamMapping {
	        { rvId 1002, ecuId 17 }
	} } }

U-PER	-getSupportedDataCallEcuFiltered (9 byte): 00 1C 82 42 03 00 00 00 22 -getSupportedDataReplyEcuFiltered (13 byte): 00 1C 83 20 28 00 00 3E A8 00 00 01 10

11.1.8 Negative access type and data parameter property filtered example

Table 16 shows a negative access filtered example for the use of the service GetSupportedData. The UGP Client requests the data parameters with the access type read-only and the data parameter property ecu-internal-signal with the getSupportedDataCallAccessTypeFiltered. In this example, the UGP Server responds with a getSupportedDataNegReply containing the errorId and the corresponding error attributes.

Table 17 — Access type and data parameter property filtered example of GetSupportedData

ASN.1	getSupportedDataCallAccessTypeFiltered UGPMessage::= {
	    callSequenceNumber 115,
	    choiceUGP getSupportedDataCall: { supportedDataFilter without-ecu-data,
	        accessType '10000'B, dataParamProperty ecu-internal-signal
	} }
	getSupportedDataNegReply UGPMessage::= { callSequenceNumber 115,
	    choiceUGP globalNegativeReply: { error {
	        { errorId 633, attribute { numeric: 32 } }
	} } }

U-PER	-getSupportedDataCallAccessTypeFiltered (6 byte): 00 1C C2 34 80 C0 -getSupportedDataNegReply (12 byte): 00 1C D4 00 B0 00 00 4F 20 20 80 20

12 UGP service cluster 4 — Data parameter access

12.1 GetValue

12.1.1 Service description

This service is used to request the values for the data parameters defined in GetSupportedData.

12.1.2 Message 'GetValueCall'

Table 18 defines the request message 'GetValueCall'.

Table 18 — Definition of the message 'GetValueCall'

<table>
<tr><td>Msg</td><td colspan="3">GetValueCall</td><td colspan="2">Request data parameter values</td></tr>
<tr><td rowspan="9">Attributes</td><td colspan="3">Name</td><td>Description</td><td>Cvt</td></tr>
<tr><td colspan="2">testInterval</td><td>condition</td><td>specifies the interpretation of the combined attributes testInterval and condition</td><td>O</td></tr>
<tr><td rowspan="4"></td><td rowspan="2">= 0</td><td>not set</td><td>reply message is sent only once</td><td rowspan="4"></td></tr>
<tr><td>set</td><td>test the condition and if test condition is true then send one reply message</td></tr>
<tr><td rowspan="2">> 0</td><td>not set</td><td>reply message is sent each time the testInterval [ms] has expired</td></tr>
<tr><td>set</td><td>test the condition each time the testInterval [ms] has expired and if test condition is true then send a reply message</td></tr>
<tr><td colspan="3">dataParamList {}</td><td>List of registered value identifiers to retrieve</td><td>O1[a]</td></tr>
<tr><td colspan="3">dataParamMapping {}</td><td>List of mappings between data parameter and ECU</td><td>O2[a]</td></tr>
<tr><td colspan="3">condition</td><td>A ComplexCondition element (see A.20). If the condition is set, the data parameter values should be responded only if the condition is true.</td><td>O</td></tr>
<tr><td colspan="6">[a] Either O1 or O2 must be defined</td></tr>
<tr><td>ASN.1</td><td colspan="5">GetValueCall::= SEQUENCE {
    testInterval   SNUM32,
    dataParamList   SEQUENCE OF Identifier   OPTIONAL,
    dataParamMapping   SEQUENCE OF DataParamMapping   OPTIONAL,
    condition   ComplexCondition   OPTIONAL,
    ...
}</td></tr>
</table>

Both attributes dataParamList and dataParamMapping define the data parameters to reply, dataParamList as a list of registered value ids, and dataParamMapping as mappings between data parameter and ECU. Either dataParamList or dataParamMapping must be defined.

The attribute condition defines under which conditions the data should be replied. A condition is defined by a simpleParam, a simpleDtc, an AND or an OR condition. A simpleParam defines a data parameter of an ECU, a compare operator, and a value. A simpleDtc defines the DTC's base and symptom identifiers and its complementary mask.

The condition is checked by the V-ITS-SG. If it is true, the requested data parameters are responded by a filled GetValueReply (see 12.1.3).

If the condition is false, the GetValueReply is empty on the first call. If the refreshInterval is greater than 0, the condition is checked by the V-ITS-SG all refreshInterval milliseconds. If the condition is true, the reply is responded as if no condition has been set. If the condition is false, no reply is responded in this time interval.

12.1.3 Message 'GetValueReply'

Table 19 defines the positive response message GetValueReply. It contains a list of the requested data parameter values defined by ecuId, rvId and a value. Dependent on the data parameter declaration (see 11.1.3), the value is a numeric, lnumeric, string, enumString, bitString, structureMissing, array, monitor, or error. See ISO 15031-5 and SAE J1979-DA for the definition of a monitor value. See 12.1.7 for a monitor example.

Table 19 — Definition of the message 'GetValueReply'

Msg	GetValueReply	Replies the `GetValueCall` by returning the values of the data parameters.	
Attributes	**Name**	**Description**	**Cvt**
	`valueTS {}`	List of `DataParamValueTS` (data parameter value time stamps) elements as decribed in A.13.	M

ASN.1	`GetValueReply::= SEQUENCE {`
	`valueTS SEQUENCE OF DataParamValueTS,`
	`...`
	`}`		

12.1.4 Error handling

If no data parameter values can be retrieved, a `GlobalNegativeReply` (see 9.2) is returned. If at least one data parameter value is available but others cannot be retrieved, the `GetValueReply` contains a list of VIErrorValues.

12.1.5 Single get data parameter list example

An example for the `GetValue` using data parameter lists is included in Table 20. The UGP Client requests the data parameter values by passing a list of four registered value ids in `getValueCallInList`. The desired parameters are `Engine Control module voltage`, `Secondary air system monitoring ready`, `Engine Coolant Temperature`, and `Hardware part number` (as defined in A.6.2, A.6.3, A.6.5, A.7, A.16). The ECU address must not be included because in this example the registered value ids are unique.

The V-ITS-SG responds with the positive reply `getValueReplyInList` returning only the values of the four desired data parameters. The values are mapped in the order of the call. So the first reply value maps to the first call parameter, etc. To interpret the values, the definition must be used. The `Engine Control module voltage` is defined with `dataTypeId` 331, who has `decimalPlaces 2, unit "V", quotient 100, min 880,` and `max 1560`, i.e. the value should be displayed with two decimal places, uses the unit 'V', must be divided by 100, and has a range of `8.80` to `15.60 V`. So the `numeric 1250` must be interpreted as `Engine Control module voltage` of 12,50 V. The second value with the `rvId 7368` has the `dataTypeId 2` which is an enumeration with values 0 for 'no' and 1 for 'yes'. So `enumString 1` must be interpreted as `'Secondary air system monitoring ready = yes'`. The `numeric 873` means an `Engine Coolant Temperature` of 87,3 °C. The `error 422` means that the `Hardware part number` cannot be read.

Using this short form of data reply, the unaligned PER for the four parameters takes only 35 bytes.

Table 20 — Data parameter list example for service GetValue

ASN.1	`getValueCallInList UGPMessage::= { callSequenceNumber 192,`
	`    choiceUGP getValueCall: { dataParamList { 1002, 7368, 2341, 1123 }`
	`} }`
	`getValueReplyInList UGPMessage::= { callSequenceNumber 192,`
	`    choiceUGP getValueReply: { valueTS {`
	`        { value numeric: 1250 }, { value enumString: 1 },`
	`        { value numeric: 873 }, { value error: 422 }`
	`} } }`

U-PER	`-getValueCallInList (21 byte):`
	`00 30 04 20  24 00 00 1F  54 00 00 E6  44 00 00 49  2C 00 00 23  18`
	`-getValueReplyInList (18 byte):`
	`00 30 05 02  00 84 E2 08  00 02 02 0D  A4 54 00 00  0D 30`

12.1.6 Asynchronous conditioned example

An asynchronous conditioned example for the `GetValue` is included in Table 21. The UGP Client requests the data parameter values by passing a `refreshInterval` of 5 000 ms, three data parameters of two ECUs, and the condition ‘Engine Coolant Temperature is greater than 110,0 °C’ in `getValueCallCond`.

Instead of the `dataParamList`, a `dataParamMapping` is included here to show a different alternative.

The V-ITS-SG first responds with a positive empty reply (`getValueReplyCond1`). The reply is empty because the temperature is not as high as defined in the condition. After some time intervals of 5 000 ms, the condition is true, and the V-ITS-SG responds asynchronous with the positive `getValueReplyCondN` including the values for the requested data parameters and possible errors for invalid signal parameters. As long as the condition is true, the data are replied every 5 000 ms.

The asynchronous service can be stopped by using the `StopService` (see 9.3).

Table 21 — Condition example for service GetValue

ASN.1	`getValueCallCond UGPMessage::= { callSequenceNumber 193,`
	`    choiceUGP getValueCall: { testInterval 5000, dataParamMapping {`
	`        { rvId 1002, ecuId 17 },`
	`        { rvId 7368, ecuId 17 },`
	`        { rvId 2341, ecuId 51 }`
	`    }, condition {`
	`        paramCond: { rvId 2341, operator gt, value numeric: 1100 }`
	`} }`
	`getValueReplyCond1 UGPMessage::= { callSequenceNumber 193,`
	`    choiceUGP getValueReply: { valueTS {`
	`} } }`
	`getValueReplyCondN UGPMessage::= { callSequenceNumber 193,`
	`    choiceUGP getValueReply: { valueTS {`
	`        { value numeric: 1250 }, { value enumString: 1 }, { value numeric: 1101 }`
	`} } }`

U-PER	`-getValueCallCond (42 byte):`
	`00 30 44 5C  00 00 9C 40  19 00 00 07  D5 00 00 00  22 40 00 0E  64 40 00 00  08 90 00 01  24 B0 00 00  06 62 20 00  02 49 48 10  89 80`
	
	`-getValueReplyCond1 (5 byte):`
	`00 30 45 00  00`
	
	`-getValueReplyCondN (13 byte):`
	`00 30 45 01  80 84 E2 08  00 02 02 11  34`

12.1.7 Monitor example

A monitor example for the `GetValue` is included in Table 22. The UGP Client requests the data parameter values by passing a monitoring registered value id (see `getValueCallMon`). The V-ITS-SG replies with a positive reply (`getValueReplyMon`) including the monitor values for the given data parameter. The monitor value includes the test values for the `testId 1` and 5 as defined in 11.1.3. The test values are 0,365 V within a range of 0,365 until 0,365 V and 0,072 s within a range of 0 until 0,100 s.

Table 22 — Monitor example for service GetValue

ASN.1	`getValueCallMon UGPMessage::= { callSequenceNumber 194,`
	`    choiceUGP getValueCall: { dataParamList { 2344 }`
	`} }`
	`getValueReplyMon UGPMessage::= { callSequenceNumber 194,`
	`    choiceUGP getValueReply: { valueTS {`
	`        { value monitor: {`
	`            { testValue 365, testValueMin 365, testValueMax 365 },`
	`            { testValue 72, testValueMin 0, testValueMax 100 }`
	`        } }`
	`} } }`

U-PER	`-getValueCallMon (9 byte):`
	`00 30 84 20  0C 00 00 49  40`
	
	`-getValueReplyMon (31 byte):`
	`00 30 85 00  88 02 70 00  00 2D B0 00  00 2D B0 00  00 2D AE 00  00 01 22 00  00 00 02 00  00 01 90`

12.1.8 Structure example 'vehicle info'

A structure example for the `GetValue` is included in Table 23. The registered value id `20025` `'vehicle info'` has `dataType 345` which is defined as structure of some data parameters (see 12.1.8). So the UGP Client requests the vehicle info data parameter values by using the registered value id for 'vehicle info' in `getValueCallVehicleInfo`. The V-ITS-SG replies with a positive reply (`getValueReplyVehicleInfo`) including only the values for all specified vehicle info data parameters. Here, the vehicle info data are responded once because the `refreshInterval` is set to 0. The vehicle info data parameters in this example are vehicle key = 'passcar-hyundai-i30-2010', vehicle type = 'car', vehicle class = 'passenger vehicle', vehicle brand = 'Hyundai', vehicle model = 'i30', vehicle variant = 'Hatchback', vehicle powertrain = '1.6l Diesel 16v. Inj., Turbo', vehicle model year = '2010', config package type = 'basic', config package name = '2013-07-12_1', V-ITS-SG firmware version = '1.10.283', enhanced diagnostic pass-thru seed = 'dj32khdskjsah32', ECU programming pass-thru seed = 'uwaj3sa32lj20' and VIN = VHJGH11763B65I860.

Table 23 — Structure example for service GetValue

ASN.1	getValueCallVehicleInfo UGPMessage::= { callSequenceNumber 196,
	    choiceUGP getValueCall: { testInterval 0, dataParamList { 20025 }
	} }
	getValueReplyVehicleInfo UGPMessage::= { callSequenceNumber 196,
	    choiceUGP getValueReply: { valueTS {
	        { value string: "passcar-hyundai-i30-2010" }, { value enumString: 4 },
	        { value enumString: 7 }, { value displayName: "Hyundai" },
	        { value displayName: "i30" }, { value displayName: "Hatchback" },
	        { value displayName: "1.6l Diesel 16v. Inj., Turbo" },
	        { value string: "2010" }, { value enumString: 0 },
	        { value string: "2013-07-12_1" }, { value string: "1.10.283" },
	        { value string: "dj32khdskjsah32" }, { value string: "uwa-j3sa32lj20" },
	        { value string: "VHJGH11763B65I860" }
	} } }

U-PER	-getValueCallVehicleInfo (9 byte): 00 31 04 20 0C 00 02 71 C8 -getValueReplyVehicleInfo (180 byte): 00 31 05 07 02 18 E1 87 9F 3C 78 79 2D D1 E7 AE EC 98 74 AD D2 CD 82 D6 4C 18 B0 08 00 08 10 00 1C 19 80 00 2A F8 07 48 79 75 6E 64 61 69 06 60 00 0A BE 40 DA 4C CC 01 98 00 02 AF A0 94 86 17 46 36 86 26 16 36 B0 66 00 00 AB EC 70 C4 B8 D9 B0 81 11 A5 95 CD 95 B0 80 C4 D9 D8 B8 81 25 B9 A8 B8 B0 81 51 D5 C9 89 BC 10 23 26 0C 58 04 00 00 04 18 C9 83 16 6B 58 37 5A C5 95 F6 20 82 18 AE 62 C1 73 27 0C C1 07 E4 D4 CD 96 BD 19 39 EB D5 CE 0E 86 6C 81 06 F5 EF 87 53 3E 78 59 B2 D9 A9 93 00 42 35 A4 4A 8F 21 8B 16 ED 99 C2 6C D6 4B 86 CC 00

12.2 SetValue

12.2.1 Service description

This service is used to set data parameter values for the data parameters defined in `SetValue`.

12.2.2 Message 'SetValueCall'

Table 24 defines the request message `SetValueCall` to set one or more data parameter values. See 12.1.3 for the data parameter value definition.

Table 24 — Definition of the message 'SetValueCall'

Msg	SetValueCall	Set data parameter values	
Attributes	**Name**	**Description**	**Cvt**
	`valueMapping {}`	List of `DataParamValueMapping` elements as defined in A.14.	M

ASN.1	`SetValueCall::= SEQUENCE {`
	`    valueMapping         SEQUENCE OF DataParamValueMapping,`
	`    ...`
	`}`		

12.2.3 Positive reply

A positive reply is handled by a `GlobalPositiveReply` (see 9.1). For an example, see 12.2.5.

12.2.4 Error handling

If a data parameter cannot be set, a `GlobalNegativeReply` (see 9.2) is returned.

12.2.5 Example

An example for the `SetValue` is included in Table 25. The UGP Client sets the data parameter values by calling the request including registered value id (rvId), optional ECU address, and data parameter value (`setValueCall`). Here the `Secondary air system monitoring ready` is set to `'no'`.

The V-ITS-SG replies with a positive reply (`setValueReplyPos`) with no additional information or a negative reply (`setValueReplyNeg`) containing a list of errors. Here a write error is replied.

Table 25 — Example for service SetValue

ASN.1	`setValueCall UGPMessage::= { callSequenceNumber 208,`
	`    choiceUGP setValueCall: { valueMapping {`
	`        { rvId 7368, ecuId 17, value enumString: 0 }`
	`} } }`
	`setValueReplyPos UGPMessage::= { callSequenceNumber 208,`
	`    choiceUGP globalPositiveReply: {`
	`} }`
	`setValueReplyNeg UGPMessage::= { callSequenceNumber 209,`
	`    choiceUGP globalNegativeReply: { error {`
	`        { errorId 714, attribute { numeric: 7368 } }`
	`} } }`

U-PER	`-setValueCall (16 byte):`
	`00 34 06 00  A8 00 01 CC  88 00 00 01  12 00 00 00`
	
	`-setValueReplyPos (4 byte):`
	`00 34 13 00`
	
	`-setValueReplyNeg (12 byte):`
	`00 34 54 00  B0 00 00 59  40 20 9C C8`

12.3 ControlValue

12.3.1 Service description

This service is used to request the control over a control function.

12.3.2 Message ‘ControlValueCall’

Table 26 defines the request message `ControlValueCall`.

Table 26 — Definition of the message ‘ControlValueCall’

Msg	ControlValueCall		Request to control a control function	
Attributes	**Name**		**Description**	**Cvt**
	`testInterval`	`= 0`	reply message is sent only once (synchronous), when the control function completes (`status = pass, fail`)	O
		`> 0`	reply message with status `in-progress` is sent each time the testInterval [ms] has expired. If the service has completed, the last reply message's status contains the result of the control-function test (`pass, fail`).	
	`dataParamList {}`		List of registered value identifiers of the control-functions to retrieve	O1[a]
	`dataParamMapping {}`		List of mappings between data parameter and ECU (see 12.1.2) defining the control-functions to start/stop on the specified ECU.	O2[a]
	`value {}`		List of `DataParamValue` elements as defined in A.11.	M
	`execute`		Type of the execution with values `start` and `stop` to start or stop the control function	M

[a] Either O1 or O2 must be defined.

ASN.1

```
ControlValueCall::= SEQUENCE {
    testInterval         SNUM32                          DEFAULT
0,
    dataParamList        SEQUENCE OF Identifier          OPTIONAL,
    dataParamMapping     SEQUENCE OF DataParamMapping
OPTIONAL,
    value                SEQUENCE OF DataParamValue,
    execute              ExecutionType,
    ...
}
ExecutionType::= ENUMERATED { start, stop, ...
}
```


12.3.3 Message ‘ControlValueReply’

Table 27 defines the positive response message `ControlValueReply`.

Table 27 — Definition of the message ‘ControlValueReply’

Msg	**ControlValueReply**	Replies the `ControlValueCall` by returning the control function`s outgoing parameter values	
Attributes	**Name**	**Description**	**Cvt**
	`status`	Execution status of the control function with the following values: `in-progress`, `pass`, `fail`.	
	`value`	List of `DataParamValue` elements representing the values with time stamp of the control function`s outgoing parameters defined with `accessType = read-only or read-write`.	M

ASN.1	`ControlValueReply::= SEQUENCE {`
	`status ExecutionStatus,`
	`value SEQUENCE OF DataParamValue,`
	`...`
	`}`		

12.3.4 Error handling

If the control function cannot be started or stopped, a `GlobalNegativeReply` (see 9.2) is returned.

12.3.5 Example 1

The UGP client performs a `ControlValue` service in the UGP server in order to command a UDS *InputOutputControlByIdentifier* service to be executed in an ECU. The In-Vehicle-Network has implemented the appropriate ISO 14229-1 UDS diagnostic protocol service. Table 28 shows an extract of the ISO 14229-1 UDS defined InputOutputControlByIdentifier service for the data identifier Air Inlet Door Position and the control option “shortTermAdjustment” on the left side. The Air Inlet Door Position should be set to 60 %.

Table 28 — ISO 14229-1 UDS InputOutputControlByIdentifier service example

ISO 14229-1 **UDS InputOutputControlByIdentifier service**		ISO 13185-1 **UGP ControlValue service**	
Description (all values are in hexadecimal)	**Value**	**rvId**	**ASN.1 dataType**
InputOutputControlByIdentifier Request SID	2F$_{16}$	FA002F00$_{16}$	structure { paramIn {
dataIdentifier: Air Inlet Door Position	9B00$_{16}$		FA002F01$_{16}$,
controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	03$_{16}$		FA002F02$_{16}$,
controlOptionRecord [controlState] = 60%	3C$_{16}$		FA002F01$_{16}$
InputOutputControlByIdentifier Response SID	6F$_{16}$		}, paramOut {
dataIdentifier: Air Inlet Door Position	9B00$_{16}$		
controlOptionRecord [inputOutputControlParameter] = shortTermAdjustment	03$_{16}$		
controlStatusRecord [controlState] = 12%	0C$_{16}$		FA002F01$_{16}$ } }

On the right side, a mapping to the VIDF Configuration definition of the InputOutputControlByIdentifier is given, which is of data type structure containing three input and one output parameters. A detailed VIDF configuration needed for the implementation of the UDS service is given in Table 29. The Data Parameter InputOutputControlByIdentifier (rvId = FA002F00$_{16}$) references the dataType 339 of type `structure`. The `structure` contains a parameter list for the input (`paramIn`) and output (`paramOut`) parameters with the rvIds of the parameters. The input parameter FA002F01$_{16}$ defines the dataIdentifier (dataType

340, lnumeric value representing an rvId), FA002F02$_{16}$ the controlOptionRecord[inputOutputControl] (dataType 341, representing an enumeration of returnControlToECU, resetToDefault, freezeCurrentState, shortTermAdjustment). The third parameter is identical to the first parameter. So the parameters content is interpreted and its value must be defined in the call. The output parameter also contains the data identifier, so the value of the data identifier is content of the reply.

Table 29 — ControlValue example configuration for UDS service InputOutputControlByIdentifier

VIDFConfig extract	`dataParam {`
	`  { rvId FA002F00, name { textId 10135, longname "InputOutputControl-`
	`ByIdentifier" },`
	`    dataTypeId 339, accessType '00100'B, dataParamProperty actuator },`
	`  { rvId FA002F01, name { textId 10136, longname "Data Identifier" },`
	`    dataTypeId 340, accessType '01000'B, dataParamProperty con-`
	`trol-value-reserved },`
	`  { rvId FA002F02, name { textId 10137, longname "Input Output Con-`
	`trol" },`
	`    dataTypeId 341, accessType '10000'B, dataParamProperty con-`
	`trol-value-reserved },`
	`  { rvId FD009B00, name { textId 10138, longname "Air Inlet Door Posi-`
	`tion" },`
	`    dataTypeId 342, accessType '11000'B, dataParamProperty ecu-inter-`
	`nal-monitor }`
	`},`
	`dataType {`
	`  { dataTypeId 339, name { textId 10139, longname "InputOutputControl-`
	`ByIdentifier"},`
	`    type structure: {`
	`      paramIn { FA002F01, FA002F02, FA002F01 }, paramOut { FA002F01 }`
	`} },`
	`  { dataTypeId 340, name { textId 10140, shortname "rvId",`
	`    longname "registered value identifier" }, type lnumeric: { unitId 0`
	`} },`
	`  { dataTypeId 341, name { textId 10141, longname "IOControl ()" },`
	`    type enumString: {`
	`      { value 0, name { textId 10142, longname "returnControlToECU" }`
	`},`
	`      { value 1, name { textId 10143, longname "resetToDefault" } },`
	`      { value 2, name { textId 10144, longname "freezeCurrentState" }`
	`},`
	`      { value 3, name { textId 10145, longname "shortTermAdjustment" }`
	`}`
	`  } },`
	`  { dataTypeId 342, name { textId 10146, longname "percent" },`
	`    type numeric: { unitId 25, min 0, max 100 } }`
	`}`

Figure 9 shows a graphical representation of this configuration.

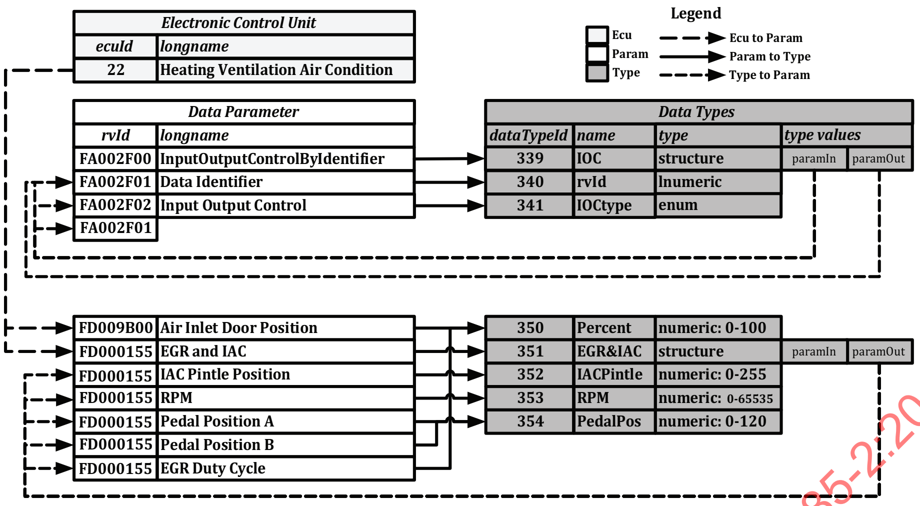


Figure 9 — ControlValue configuration dependencies

It contains additional parameters use in a second example.

Table 30 defines the ControlValue runtime example for the UDS service *InputOutputControlByIdentifier*. The `controlValueCall1` contains a `dataParamMapping` and some `values`. The `dataParamMapping` uses `rvId` = $FA002F00_{16}$ and `ecuId` = 22, i.e. use the UDS service *InputOutputControlByIdentifier* on the *Heating Ventilation Air Condition*. The `lnumeric` value $FD009B00_{16}$ references the *Air Inlet Door Position*. The `enumString` 3 means *shortTermAdjustment*. The third input parameter of the *InputOutputControlByIdentifier* references the data identifier again, so use its value. The numeric value 60 means: Set the value of the *Air Inlet Door Position* to 60 %.

Table 30 — ControlValue example InputOutputControlByIdentifier – Air Inlet Door Position

ASN.1	`controlValueCall1 UGPMessage::= { callSequenceNumber 348,`
	`    choiceUGP controlValueCall: { testInterval 0, dataParamMapping {`
	`        { rvId fa002f00, ecuId 22 }`
	`    }, value {`
	`        lnumeric: fd009b00, enumString: 3, numeric: 60`
	`    }, execute start`
	`} }`
	`------------------------------------------`
	`controlValueReply1 UGPMessage::= { callSequenceNumber 348,`
	`    choiceUGP controlValueReply: { status pass, value {`
	`        numeric: 10`
	`    }`
	`} }`

U-PER	`--controlValueCall1 (24 byte):`
	`00 57 07 10  11 E8 00 BC  02 00 00 00  58 0C 2F A0  13 60 04 00  03 04 01 E0`
	
	`--controlValueReply1 (8 byte):`
	`00 57 08 10  10 40 05 00`

The corresponding (callSequenceNumber is equal) `controlValueReply` returns with the `status pass` and the `numeric` value 10, i.e. the temporary result state of the *Air Inlet Door Position* at the time of the execution of the UDS service is 10 %. If the UGP client implementation requires a continuous refresh of the status of the Air Inlet Door Position, it is recommended to send a `GetValueCall` with the same `rvId` with an appropriate `testInterval`.

12.3.6 Example 2

A second example uses the same UDS service with a data identifier containing more than one parameter. Figure 9 contains the graphical representation of the VIDF Configuration of the data parameter *EGR and IAC*. It is defined as structure with the sub parameters *IAC Pintle Position*, *RPM*, *Pedal Position A*, *Pedal Position B* and *EGR Duty Cycle*. Table 31 shows the executed UGP services to set the Pintle Position and get all EGR and IAC parameters. The `controlValueCall2` contains the `rvId` = FA002F00$_{16}$ and `ecuId` = 22, i.e. use the UDS service *InputOutputControlByIdentifier* on the *Heating Ventilation Air Condition*. The `lnumeric` value FD000155$_{16}$ references the *EGR and IAC*. The `enumString` 3 means *shortTermAdjustment*. The residual parameters identify the sub parameters of the *EGR and IAC*. The `numeric` value 7 means set the *IAC Pintle Position* to 7. An error value 0 as input parameter means this parameter cannot be changed. To adjust the EGR Duty Cycle, e.g. the last parameter must be set to a numeric value.

The corresponding `controlValueReply2` returns the current values of all sub parameters: IAC Pintle Position = 7, RPM = 750 rpm, Pedal Position A = 8 %, Pedal Position B = 16 %, EGR Duty Cycle = 35 %.

Table 31 — ControlValue example InputOutputControlByIdentifier – EGR and IAC

ASN.1	`controlValueCall2 UGPMessage::= { callSequenceNumber 349,`
	`    choiceUGP controlValueCall: { testInterval 0, dataParamMapping {`
	`        { rvId fa002f00, ecuId 22 }`
	`    }, value {`
	`        lnumeric: fd000155, enumString: 3, numeric: 7,`
	`        error: 0, error: 0, error: 0, error: 0`
	`    }, execute start`
	`} }`
	`controlValueReply2 UGPMessage::= { callSequenceNumber 349,`
	`    choiceUGP controlValueReply: { status pass, value {`
	`        numeric: 7, numeric: 750, numeric: 8, numeric: 16, numeric: 35`
	`    }`
	`} }`

U-PER	`--controlValueCall2 (43 byte):`
	`00 57 47 10  11 E8 00 BC  02 00 00 00  58 1C 2F A0  00 2A A4 00  03 04 00 3A`
	`A0 00 00 00  15 00 00 00  00 A8 00 00  00 05 40 00  00 00 00`
	
	`--controlValueReply2 (18 byte):`
	`00 57 48 10  50 40 03 82  0B B8 10 01  00 80 10 04  01 18`

13 UGP service cluster 5 — Diagnostic trouble code information access

13.1 GetDtcInfo

13.1.1 Service description

This service is used to request the current diagnostic trouble codes (DTCs) of the vehicle.

13.1.2 Message 'GetDtcInfoCall'

Table 32 defines the request message `GetDtcInfoCall`. If no attribute is defined, the call requests all DTCs of all ECUs once without DTC environment data.

The attribute `testInterval` defines the time interval between the replies by the V-ITS-SG. With the attribute `ecuList`, the DTCs can be filtered to the specified ECUs. The attribute `withEnvData` set to TRUE requests the environment data for the DTCs.

Table 32 — Definition of the message "GetDtcInfoCall'

Msg	GetDtcInfoCall			Request the current DTCs	
Attributes	**Name**			**Description**	**Cvt**
	`test-Interval`		`condition`	specifies the interpretation of the combined attributes `testInterval` and `condition`	O
		`= 0`	not set	reply is sent once synchronous (default value)	
			set	reply is sent once and contains DTCs, if a DTC occurred and the condition is TRUE	
		`> 0`	not set	reply message is sent each time the testInterval [ms] has expired	
			set	test the condition each time the testInterval [ms] has expired and if test condition is true then send a reply message	
	`rdtcBaseId`			Filters the reply th the defined DTC base identifier	O
	`rdtcSymptomId`			Filters the reply th the defined DTC symptom identifier	O
	`ecuList {}`			List of the addresses of the in-vehicle network ECUs from where the DTCs should be retrieved	O
	`withEnvData`			FALSE: retrieve no environment data (default value) TRUE: retrieve environment data	O
	`condition`			A `ComplexCondition` element (see A.20). If the condition is set, the data parameter values should be responded only if the condition is true.	O

ASN.1	`GetDtcInfoCall::= SEQUENCE {`
	`testInterval SNUM32 DEFAULT 0,`
	`rdtcBaseId UNUM16 OPTIONAL,`
	`rdtcSymptomId UNUM16 OPTIONAL,`
	`ecuList SEQUENCE OF Identifier OPTIONAL,`
	`withEnvData BOOLEAN DEFAULT FALSE,`
	`condition ComplexCondition OPTIONAL,`
	`...`
	`}`				

13.1.3 Message 'GetDtcInfoReply'

Table 33 defines the positive response message `GetDtcInfoReply` containing a list of DTC information (see A.15) including `rtcBaseId`, `rtcSymptomId`, optionally `ecuId` and complementary like `testFailed`, `pendingDTC` or `permanent`. If the `GetDtcInfoCall` has set the `withEnvData = TRUE`, the environment data are included in the DTC information.

Table 33 — Definition of the message "GetDtcInfoReply'

Msg	GetDtcInfoReply	Replies the `GetDtcInfoCall` by returning DtcInfo elements.	
Attrib.	**Name**	**Description**	**Cvt**
	`dtcInfo {}`	List of DtcInfo elements (see A.15)	M

Table 33 *(continued)*

ASN.1	`GetDtcInfoReply::= SEQUENCE {`
	`    dtcInfo            SEQUENCE OF DtcInfo            OPTIONAL,`
	`...`
	`}`

13.1.4 Error handling

If the DTC information cannot be retrieved, a `GlobalNegativeReply` (see 9.2) is returned.

13.1.5 Examples

Table 34 shows typical examples for the use of the service `GetDtcInfo`.

Table 34 — Examples for service 'GetDtcInfo

ASN.1

```
getDtcInfoCall0 UGPMessage::= { callSequenceNumber 256,
    choiceUGP getDtcInfoCall: { ecuList { 32 }
} }
getDtcInfoReply0 UGPMessage::= { callSequenceNumber 256,
    choiceUGP getDtcInfoReply: {
} }
getDtcInfoCall1 UGPMessage::= { callSequenceNumber 257,
    choiceUGP getDtcInfoCall: { testInterval 10000
} }
getDtcInfoReply1 UGPMessage::= { callSequenceNumber 257,
    choiceUGP getDtcInfoReply: { dtcInfo {
        { rDtcBaseId 5, rDtcSymptomId 1, ecuId 21,
          complementary '0010000000000000'B },
        { rDtcBaseId 256, rDtcSymptomId 1, ecuId 17,
          complementary '0000000100000000'B },
        { rDtcBaseId 157, rDtcSymptomId 1, ecuId 17,
          complementary '0010000000100000'B }
} } }
getDtcInfoCall2 UGPMessage::= { callSequenceNumber 258,
    choiceUGP getDtcInfoCall: { ecuList { 21 }, withEnvData TRUE
} }
getDtcInfoReply2 UGPMessage::= { callSequenceNumber 258,
    choiceUGP getDtcInfoReply: { dtcInfo {
        { rDtcBaseId 5, rDtcSymptomId 1, ecuId 21,
          complementary '0010000000000000'B, envData {
            { value numeric: 1250 }, { value numeric: 910 } } }
} } }
```

The UGP Client requests the DTC with `getDtcInformationCall0` for a single ECU `32`. If there are no DTCs, so `getDtcInfoReply0` is empty.

getDtcInfoCall1 requests asynchronous DTCs using a testInterval of 10 s. If DTC information is available at this time, the getDtcInfoReply1 replies with the requested DTC information including rDtcBaseId, rDtcSymptomId, ecuId and complementary.

getDtcInfoCall2 requests DTCs for a single ECU with environment data. getDtcInfoReply2 replies with the DTCs and its environment data if it exists. The values of the environment data must be mapped to the DTC base (see A.9) and DTC symptom (see A.10) definition.

13.2 ClearDtcInfo

13.2.1 Service description

This service is used to request to clear the vehicles DTCs and information.

13.2.2 Message 'ClearDtcInfoCall'

Table 35 defines the request message ClearDtcInfoCall.

Table 35 — Definition of the message 'ClearDtcInfoCall'

Msg	ClearDtcInfoCall	Request to clear the DTCs of the defined ECUs	
Attributes	**Name**	**Description**	**Cvt**
	ecuList {}	List of identifiers of the in-vehicle network ECUs to clear the DTCs. If no list is present, the DTCs of all ECUs should be cleared.	O

ASN.1	ClearDtcInfoCall::= SEQUENCE {
	ecuList SEQUENCE OF Identifier OPTIONAL,
	...
	}		

13.2.3 Positive reply

A positive reply is handled by a GlobalPositiveReply (see 9.1). For an example, see 12.2.5.

13.2.4 Error handling

If the data information cannot be cleared, a GlobalNegativeReply (see 9.2) is returned.

13.2.5 Example

An example for the ClearDtcInfo service is included in Table 36. The UGP Client requests the clearing of all DTCs by calling clearDtcInfoCall0. The V-ITS-SG replies with a positive clearDtcInfoPosReply with no additional information. The clearDtcInfoCall1 requests the clearing of the DTCs of a single ECU. The V-ITS-SG responds with a negative clearDtcInfoNegReply containing the unified gateway protocol error describing the problem.

Table 36 — Example for service ClearDtcInfo

ASN.1	`clearDtcInfoCall0 UGPMessage::= { callSequenceNumber 272,`
	`    choiceUGP clearDtcInfoCall: {`
	`} }`
	`clearDtcInfoPosReply UGPMessage::= { callSequenceNumber 272,`
	`    choiceUGP globalPositiveReply: {`
	`} }`
	`clearDtcInfoCall1 UGPMessage::= { callSequenceNumber 273,`
	`    choiceUGP clearDtcInfoCall: { ecuList { 206 }`
	`} }`
	`clearDtcInfoNegReply UGPMessage::= { callSequenceNumber 273,`
	`    choiceUGP globalNegativeReply: { error {`
	`        { errorId 633, attribute { numeric: 32 } }`
	`} } }`

U-PER	`--clearDtcInfoCall0 (4 byte):`
	`00 44 0B 00`
	
	`--clearDtcInfoPosReply (4 byte):`
	`00 44 13 00`
	
	`--clearDtcInfoCall1 (9 byte):`
	`00 44 4B 40  60 00 00 33  80`
	
	`--clearDtcInfoNegReply (12 byte):`
	`00 44 54 00  B0 00 00 4F  20 20 80 20`

14 UGP service cluster 6 — In-vehicle network access

14.1 EnablePassThru

14.1.1 Service description

This `EnablePassThru` service is used to request 'pass thru' access for the e.g. vehicle manufacturer diagnostic and ECU memory programming protocol. A seed-key mechanism shall be implemented to grant access to utilize the `EnablePassThru` service.

14.1.2 Message 'EnablePassThruCall'

Table 37 defines the request message `EnablePassThruCall` to request pass-thru access to in-vehicle network ECUs to perform diagnostics and ECU memory programming.

Table 37 — Definition of the message 'EnablePassThruCall'

Msg	**EnablePassThruCall**	Request to enable or disable the pass thru modus.	
Attributes	**Name**	**Description**	**Cvt**
	`label`	Label of the pass thru seed to enable or disable the pass thru	M

Table 37 *(continued)*

Msg	EnablePassThruCall	Request to enable or disable the pass thru modus.	
	`key`	Key to enable the pass thru; if the `key` lacks, the pass thru will be disabled	O

ASN.1	`EnablePassThruCall::= SEQUENCE {`
	`label String,`
	`key String OPTIONAL`
	`}`		

The seed label of the activities shall be retrieved by getting the value of the corresponding data parameter. The vehicle info data parameter collection should contain seeds (see 12.1.8) for authorized P-ITS-Ss. The example in 12.1.8 responds pass-thru seeds with label and value for enhanced diagnostic (label "`ENH_DIAG_PASS_THRU_SEED`") and ECU programming (label "`ECU_PRG_PASS_THRU_SEED`"). Label and key must be added as attributes to enable the pass-thru of the corresponding activity. To disable the activity, the `key` must be removed as attribute.

14.1.3 Positive reply

A positive reply is handled by a `GlobalPositiveReply` (see 9.1).

14.1.4 Error handling

If the enabling or disabling of the pass-thru fails, a `GlobalNegativeReply` (see 9.2) is returned.

14.1.5 Example

An example for the `EnablePassThruCall` is included in Table 38. The UGP Client enables the pass-thru using the label and a key from the GetValue 'vehicle info' (see 12.1.8) with the `enablePassThruCall`. The V-ITS-SG replies with a global positive reply `enablePassThruReply`. To disable the pass-thru, the UGP Client has to use the request without key (`disablePassThruCall`).

Table 38 — Example for service EnablePassThru

ASN.1	`enablePassThruCall UGPMessage::= { callSequenceNumber 332,`
	`    choiceUGP enablePassThruCall: {`
	`        label "ENH_DIAG_PASS_THRU_SEED", key "flkja0932jdla9323jddff3d"`
	`} }`
	`enablePassThruReply UGPMessage::= { callSequenceNumber 332,`
	`    choiceUGP globalPositiveReply: {`
	`} }`
	`disablePassThruCall UGPMessage::= { callSequenceNumber 333,`
	`    choiceUGP enablePassThruCall: {`
	`        label "ECU_PRG_PASS_THRU_SEED"`
	`} }`

U-PER	`--enablePassThruCall (47 byte):`
	`00 53 0C 45  E2 CE 91 7E  24 98 31 EF  D0 83 4E 9D  FA 92 29 55  BF 4E 2C 58`
	`83 19 B6 6B  D5 85 83 96  6C B5 64 D9  85 CB 36 4C  F5 64 C9 9B  33 3C 80`
	
	`--enablePassThruReply (4 byte):`
	`00 53 13 00`
	
	`--disablePassThruCall (24 byte):`
	`00 53 4C 05  A2 C3 AB 7E  85 28 F7 E8  41 A7 4E FD  49 14 AA DF  A7 16 2C 40`

15 UGP service cluster 7 — Maintenance

15.1 ListFile

15.1.1 Service description

This service is used to request a download of core, configuration, or log files from the V-ITS-SG.

15.1.2 Message 'ListFileCall'

Table 39 defines the request message `ListFileCall` to retrieve a list of all files of the given `fileType` (`core, configuration, log, snapshot, calibration`).

Table 39 — Definition of the message 'ListFileCall'

Msg	**ListFileCall**	Request a list of files of the given fileType from the V-ITS-SG.	
Attributes	**Name**	**Description**	**Cvt**
	`fileType`	Type of the file: `core`, `configuration, log, snapshot.`	M

Table 39 *(continued)*

Msg	ListFileCall	Request a list of files of the given fileType from the V-ITS-SG.

ASN.1	`ListFileCall::= SEQUENCE {`
	`    fileType          FileType,`
	`    ...`
	`}`
	`FileType::= ENUMERATED { core, configuration, log, snapshot, calibra-tion, ...`
	`}`	

15.1.3 Message 'ListFileReply'

Table 40 defines the positive response message `ListFileReply`.

Table 40 — Definition of the message 'ListFileReply'

Msg	ListFileReply	Replies the `ListFileCall` by returning a list of corresponding file names.	
Attributes	**Name**	**Description**	**Cvt**
	`fileInfo {}`	A list of file information parameters names of the fileType defined in the corresponding call including following sub attributes:	M
	– `filename`	— Name of the file	M
	– `version`	— Version of the file if exists (e.g. core and configuration)	O

ASN.1	`ListFileReply::= SEQUENCE {`
	`    fileInfo          SEQUENCE OF FileInfo,`
	`    ...`
	`}`
	`FileInfo::= SEQUENCE {`
	`    fileName          String,`
	`    version            String`
	`OPTIONAL,`
	`    ...`
	`}`		

15.1.4 Error handling

If no file directory can be listed, a `GlobalNegativeReply` (see 9.2) is returned.

15.1.5 Example

Table 41 shows an example for the use of the service `ListFile`. The UGP Client requests the directory listing with `listFileCall`. The UGP Server responds with a `listFilesReply` containing a list of file information containing the file name and its version.

Table 41 — Example for service ListFile

ASN.1	`listFileCall UGPMessage::= { callSequenceNumber 448,`
	`    choiceUGP listFileCall: { fileType configuration`
	`} }`
	`listFileReply UGPMessage::= { callSequenceNumber 448,`
	`    choiceUGP listFileReply: { fileInfo {`
	`        { fileName basic.cfgvidfg, version 2013071601 },`
	`        { fileName saej1979-da.cfgvidfg, version 2013072401 },`
	`        { fileName passcar_hyundai_i30_2007.cfgvidfm, version 2014090402 }`
	`} } }`

U-PER	`--listFileCall (4 byte):`
	`00 70 0D 10`
	`--listFileReply (96 byte):`
	`00 70 0E 01  A1 D8 B0 F3  D3 8D 76 3C  D9 FB 69 C9  9B 38 53 26  0C 59 B0 6E`
	`C5 B3 06 28  A7 3C 39 75  31 72 DD CA  DC 98 57 63  CD 9F B6 9C  99 B3 85 32`
	`60 C5 9B 06  EC 9A 30 62  90 F0 C3 CF  9E 3C 3C AF  E8 F3 D7 76  4C 3A 6F E9`
	`66 C2 FB 26  0C 1B AE C7  9B 3F 6D 39  33 6D 0A 64  C1 8B 46 0E  58 34 60 C8`

15.2 ManageFile

15.2.1 Service description

This service requests a file for upload.

15.2.2 Message ‘ManageFileCall’

Table 42 defines the request message `ManageFileCall` using `activityType`, `fileType`, `fileName`, `fileSize`, `data` and `crc`.

Table 42 — Definition of the message ‘ManageFileCall’

Msg	**ManageFileCall**	Request upload of a file	
Attributes	**Name**	**Description**	**Cvt**
	`activityType`	Type of the activity: `upload`, `download`, `delete`	M
	`fileType`	Type of the file: `core`, `configuration`, `log`, `snapshot`, `calibration`	M
	`fileName`	File name on V-ITS-SG	M
	`fileSize`	Size of the file in bytes to be uploaded	C[a]
	`data`	The file data as Octet String	C[a]
	`crc`	Cyclic redundancy check value	C[a]
[a] Parameters are only used if `activityType = upload`.			

Table 42 *(continued)*

Msg	ManageFileCall	Request upload of a file

ASN.1	`ManageFileCall::= SEQUENCE {`
	`    activityType          FileActivityType,`
	`    fileType              FileType,`
	`    fileName              String,`
	`    fileSize              SNUM32                                    OPTIONAL,`
	`    data                  OctetString`
	`OPTIONAL,`
	`    crc                   SNUM32`
	`OPTIONAL,`
	`    ...`
	`}`
	`FileActivityType::= ENUMERATED { upload, download, delete, ...`
	`}`	

15.2.3 Message 'ManageFileReply'

Table 40 defines the positive response message `ManageFileReply`.

Table 43 — Definition of the message 'ManageFileReply'

Msg	ManageFileReply	Replies the `ManageFileCall`.	
Attributes	**Name**	**Description**	**Cvt**
	`filesize`	The size of the file in bytes	C[a]
	`data`	The file data as Octet String	C[a]
	`crc`	Cyclic redundancy check value	C[a]
[a] Parameters are only used if `activityType = download`.			

ASN.1	`ManageFileReply::= SEQUENCE {`
	`    fileSize              SNUM32                                    OPTIONAL,`
	`    data                  OctetString`
	`OPTIONAL,`
	`    crc                   SNUM32`
	`OPTIONAL,`
	`    ...`
	`}`		

The `ManageFileReply` returns the `filesize` in bytes, the `data` and a `crc` checksum of the file if the `activityType = download`. Otherwise the `ManageFileReply` is empty.

15.2.4 Error handling

If the requested service cannot be executed, `GlobalNegativeReply` (see 9.2) is returned.

15.2.5 Example download

A download example for the `ManageFile` is included in Table 44. The UGP Client requests the download of a configuration file with `manageFileCallDownload` using the `activityType download`, the `fileType configuration` and passing the file name. The V-ITS-SG replies with a positive `manageFile-`

ReplyDownload containing the fileSize, the corresponding data as octet string and the CRC value to check the data.

Table 44 — Example for ManageFile download

ASN.1	manageFileCallDownload UGPMessage::= { callSequenceNumber 448,
	    choiceUGP manageFileCall: { activityType download, fileType configura-tion,
	        fileName "passcar_hyundai_i30_2007_2007110402"
	} }
	manageFileReplyDownload UGPMessage::= { callSequenceNumber 456,
	    choiceUGP manageFileReply: { fileSize 1234300,
	        data '3FE2EBAD471005'H, crc 432
	} }

U-PER	--manageFileCallDownload (36 byte): 00 70 0F 02 48 F8 61 E7 CF 1E 1E 57 F4 79 EB BB 26 1D 37 F4 B3 61 7D 93 06 0D EF B2 60 C1 BB 16 2C 1A 30 64 --manageFileReplyDownload (20 byte): 00 72 10 78 01 2D 57 C0 73 FE 2E BA D4 71 00 58 00 00 1B 00

15.2.6 Example upload

An upload example for the ManageFile is included in Table 45. The UGP Client sends a configuration file by calling the request manageFileCallUpload using the fileType configuration and adding the name, size, data, and CRC of the file. The V-ITS-SG replies with a positive manageFileReplyUpload with no additional information.

Table 45 — Example for ManageFile upload

ASN.1	manageFileCallUpload UGPMessage::= { callSequenceNumber 464,
	    choiceUGP manageFileCall: { activityType upload, fileType configura-tion,
	        fileName "passcar_hyundai_i30_2007_2007110402", fileSize 2155109,
	        data '3FE2EBAD47100744545113422434265651234234122342342404'H, crc 614
	} }
	manageFileReplyUpload UGPMessage::= { callSequenceNumber 464,
	    choiceUGP manageFileReply: {
	} }

U-PER	--manageFileCallUpload (71 byte): 00 74 0F 70 48 F8 61 E7 CF 1E 1E 57 F4 79 EB BB 26 1D 37 F4 B3 61 7D 93 06 0D EF B2 60 C1 BB 16 2C 1A 30 65 00 41 C4 CA 34 7F C5 D7 5A 8E 20 0E 88 A8 A2 26 84 48 68 4C AC A2 46 84 68 24 46 84 68 48 09 00 00 04 CC --manageFileReplyUpload (4 byte): 00 74 10 00

15.2.7 Example delete

A delete example for the ManageFile is included in Table 46. The UGP Client sends a deleteCall using the fileType configuration and adding only the name of the file. The V-ITS-SG replies with a positive manageFileReplyDelete with no additional information.

Table 46 — Example for ManageFile delete

ASN.1	manageFileCallDelete UGPMessage::= { callSequenceNumber 464,
	    choiceUGP manageFileCall: { activityType delete, fileType configura-tion,
	        fileName "passcar_hyundai_i30_2007_2007110402"
	} }
	manageFileReplyDelete UGPMessage::= { callSequenceNumber 464,
	    choiceUGP globalNegativeReply: { error { { errorId 38 } }
	} }

U-PER	--manageFileCallDelete (36 byte): 00 74 0F 04 48 F8 61 E7 CF 1E 1E 57 F4 79 EB BB 26 1D 37 F4 B3 61 7D 93 06 0D EF B2 60 C1 BB 16 2C 1A 30 64 --manageFileReplyDelete (9 byte): 00 74 14 00 90 00 00 04 C0

Annex A
(normative)

Vehicle Interface Data Format definition (VIDF)

A.1 VIError

Table A.1 defines the VIError.

Table A.1 — VIError definition

Attributes	Name	Description	Cvt
	`errorId`	error identifier	M
	`name`	Display name of the error	M
	`attributeCount`	number of attributes used by this error	M
Example	`{ errorId 10118, name { textId 10118, longname "Invalid file activity type" },` `attributeCount 0 }`		

A.2 UnitType

Table A.2 defines the UnitType. The UnitType collects units in groups. Typical unit types are length, weight, time, temperature, pressure, etc.

Table A.2 — UnitType definition

Attributes	Name	Description	Cvt
	`unitTypeId`	Identifier of the unit type	M
	`name`	Display name of the unit type	M

Example	`{ unitTypeId 5, name { textId 9005, longname "time" } },`
	`{ unitTypeId 7, name { textId 9007, longname "temperature" } },`
	`{ unitTypeId 8, name { textId 9008, longname "pressure" } }`		

A.3 Unit

Table A.3 defines the Unit and, its relation to a unit type, formula and its attributes to calculate the default unit inside a unit type.

Table A.3 — Unit definition

Attributes	Name	Description	Cvt
	`unitTypeId`	Identifier of the unit type as reference (see A.2)	M
	`unitId`	Identifier of the unit	M
	`Name`	Display name of the unit	M
	`formula`	used formula to calculate default unit of same unit type	M
	`c0`	C0 parameter for formula calculation	M

Table A.3 *(continued)*

	c1	C1 parameter for formula calculation	M
	c2	C2 parameter for formula calculation	M
Example	{ unitTypeId 7, unitId 11, name { textId 9111, shortname "°C", longname "degree celsius" }, Formula (0), c0 0, c1 0, c2 0 }, { unitTypeId 7, unitId 39, name { textId 9139, shortname "°F", longname "fahrenheit" }, Formula (4), c0 -32, c1 10, c2 18 }		

A.4 Provider

Table A.4 defines the Provider.

Table A.4 — Provider definition

Attributes	**Name**	**Description**	**Cvt**
	providerId	Identifier of the provider	M
	name	Display name of the provider	M
Example	{ providerId 33, name { textId 9233, longname "Honda" } }, { providerId 34, name { textId 9234, longname "Hyundai" } }		

A.5 Ecu

Table A.5 defines the Ecu.

Table A.5 — Ecu definition

Attributes	**Name**	**Description**	**Cvt**
	ecuId	Identifier of the ECU	M
	name	Display name of the ECU	M
	providerId	Identifier of a provider	M
	parentId	Identifier of a parent ECU	O
Example	{ ecuId 17, name { textId 10202, shortname "ECM", longname "engine control module" }, providerId 0 }, { ecuId 21, name { textId 10203, shortname "CTCM", longname "coolant temperature control module" }, providerId 0 }		

A.6 DataType

A.6.1 DataType attributes

Table A.6 defines the DataType attributes.

Table A.6 — DataType attributes

Attributes	**Name**	**Description**	**Cvt**
	dataTypeId	Identifier of the data type	M
	name	DisplayName of the data type (see A.17)	O
	type	Type choice between the following sub types:	M

Table A.6 *(continued)*

	— numeric	— A numeric value (Numeric) containing the following sub attributes (see A.6.2):	C0[a]
	— decimalPlaces	— Number of decimal places for the display	O
	— unitId	— Unit identifier as reference (see A.3)	M
	— factor	— Factor to multiply with; default value is 1	O
	— quotient	— Quotient to divide with; default value is 1	O
	— addend	— Addend to add to; default value is 0	O
	— min	— Minimal numeric value	O
	— max	— Maximal numeric value	O
	— lnumeric	— A long numeric value (LNumeric) containing the same attributes as numeric (see A.6.2):	C1[a]
	— string	— A limited string (LimitedString) containing the following sub attributes (see A.6.3):	C2[a]
	— allowedCharacters	— A list (regular expression) of the allowed characters	O
	— minLen	— Minimal length of the desired string	O
	— maxLen	— Maximal length of the desired string	O
	— displayName	— An internationalizable string (contains no data for definition)	C3[a]
	— enumString {}	— A list of enumeration string item values (EnumStringItem) containing (see A.6.5):	C4[a]
	— value	— Value of the enumeration string item	M
	— name	— Display name of the string table item	M
	— bitString {}	— A list of bit string item values (BitStringItem) containing the following attributes (see A.6.6):	C5[a]
	— bit	— Value of the enumeration string item	M
	— name	— Display name of the string table item	M
	— structure	— A Structure element containing following attributes (see A.6.7):	C6[a]
	— param	— A list of the contained registered value identifiers (see A.7)	M
	— convention	— Convention of the structure (mandatory, optional, conditional); default value is mandatory	O
	— array	— DataTypeId of the elements of the data array	C7[a]
	— monitor	— A list of monitor item values (MonitorItem) containing the following attributes:	C8[a]
	— testId	— Test id of the monitor item	M
	— decimalPlaces	— Number of decimal places for the display	O
	— unitId	— Unit identifier as reference (see A.3)	M
	— factor	— Factor to multiply with; default value is 1	O
	— quotient	— Quotient to divide with; default value is 1	O
	— addend	— Addend to add to; default value is 0	O
	— octet	— An octet string	C9[a]

[a] C0, C1, C2, C3, C4, C5, C6, and C7 are the choices of the dataTypes' type. Only one of C0 or C1 or C2 or C3 or C4 or C5 or C6 or C7 or C8 or C9 can be used.

The data types can be `numeric`, `lnumeric` (long numeric), `string`, `displayName`, `enumString` (an enumeration string), `bitString`, `structure`, `array`, `monitor` or `octet`.

A.6.2 numeric, lnumeric

The data type `numeric` of ASN.1 type `Numeric` is used to realize small integer and floating point values (2 bytes). The data type `lnumeric` of ASN.1 type `LNumeric` is used to realize long integer and long floating point values (4 bytes). To minimize the data parameter value length and not to transport floating values, all floating values can be defined as SNUM16 (`Numeric`) or SNUM32 (`LNumeric`). To transform into the floating value, the calculation attributes factor (default = 1), quotient (default = 1) and addend (default = 0) are used. The Calculation rule for the realValue, realMinValue and realMaxValue is defined in Formula (1). The real value is rounded to decimalPlaces with unit.

Definition of formula

`realValue = value * factor / quotient + addend` (1)

Table A.7 — Numeric example

Example	`{ dataTypeId 331, name { textId 10001, longname "voltage in 1/100 V" },`
	`type numeric: {`
	`decimalPlaces 2, unitId 17, factor 1, quotient 100, addend 0, min 880, max 1560`
	`} },`
	`{ dataTypeId 333, name { textId 10003, longname "temperature in 1/10 °C" },`
	`type numeric: {`
	`decimalPlaces 1, unitId 11, factor 1, quotient 10, addend 0, max 1200`
	`} }`

The first example of Table A.7 (`dataTypeId 331`) defines a numeric value between 8,80 V and 15,60 V (realMinValue = 880 * 1 / 100 + 0 = 8,8; realMaxValue = 1 560 * 1 / 100 + 0 = 15,6; 2 decimal places; unit is V).

The second example of Table A.7 (`dataTypeId 333`) defines a numeric value with a maximum of 120,0 °C.

A.6.3 string

The data type `string` of ASN.1 type `LimitedString` realizes limited and unlimited strings. The optional attribute `allowedCharacters` contains a regular expression of the allowed characters in the string. The optional attributes `minLen` and `maxLen` define the range of the character count. The first example in Table A.8 defines a 'Vehicle Identification Number' string with exactly 17 characters (`minLen = maxLen = 17`) and can contain only capital letters from `A-H`, `J-N`,`P`,`R-Z` or the digits `0-9`. The second example defines an unlimited string.

Table A.8 — Limited string example

Example	{ dataTypeId 360, name { textId 10058, longname " Vehicle Identification Number " },
	  type string: {
	    allowedCharacters "A..HJ..NPR..Z0..9", minLen 17, maxLen 17
	} },
	{ dataTypeId 334, name { textId 10004, longname "unlimited string" }, type string: {
	} }

A.6.4 displayName

The data type `displayName` of ASN.1 type `NULL` realizes internationalizable strings and contains no attributes. Table A.9 defines an example for DisplayName.

Table A.9 — DisplayName example

Example	{ dataTypeId 342, name { textId 10015, longname "display name" }, type displayName: NULL }

A.6.5 enumString

The data type `enumString` of ASN.1 type `SEQUENCE OF EnumStringItem` realizes enumerations of values in text form. Only one of the `EnumStringItems` can be selected. The example in Table A.10 defines an enumeration of the two texts 'no' and 'yes'. If the enum value is `0`, '`no`' is selected, if the enum value is `1`, '`yes`' is selected.

Table A.10 — Enum string example

Example	{ dataTypeId 332, name { textId 10002, longname "answer (no, yes)" },
	  type enumString: {
	    { value 0, name { textId 10059, longname "no" } },
	    { value 1, name { textId 10060, longname "yes" } }
	} }

A.6.6 bitString

The data type `bitString` of ASN.1 type `SEQUENCE OF BitStringItem` realizes combinations of enumerations of values in text form. So, any of the `BitStringItems` can be selected. The example in Table A.11 defines a weather selection matrix with bits for rain, snow, ice, fog, and strong wind. Any of the 'weather bits' can be selected.

Table A.11 — Bit string example

Example	`{ dataTypeId 33, name { textId 10329, longname "weather condition" },`
	`  type bitString: {`
	`    { bit 0, name { textId 10330, longname "rain" } },`
	`    { bit 1, name { textId 10331, longname "snow" } },`
	`    { bit 2, name { textId 10332, longname "ice" } },`
	`    { bit 3, name { textId 10333, longname "fog" } },`
	`    { bit 4, name { textId 10334, longname "strong wind" } }`
	`} }`

The general value is calculated by Formula (2).

Definition of formula

$$\text{value}=\sum_{\text{bit}=1}^{\text{maxbit}}\begin{cases}2^{\text{bit}}, & \text{bit is selected}\\ 0, & \text{bit is not selected}\end{cases} \qquad (2)$$

The value for 'snow and ice' is calculated by $\texttt{value}_{\texttt{snow\&ice}} = 2^{\texttt{bitsnow}} \oplus 2^{\texttt{bitice}} = 21 + 2^2 = 2 + 4 = 6$. So the bit string value must be coded as `bitString: 6`.

A.6.7 structure

The data type `structure` of ASN.1 type `Structure` realized summarizations of data parameters. The first example in Table A.12 defines a structure with the sub parameters 7, 8, and 9. A sub parameter can be a structure as well. Recursions are not allowed, i.e. a parameter cannot have itself as direct or indirect child.

Table A.12 — Structure example

Example	`{ dataTypeId 6, name { textId 10308, longname "vehicle motion {}" },`
	`  type structure: {`
	`    paramOut { 7, 8, 9 }, convention conditional`
	`} },`
	`{ dataTypeId 345, name { textId 10020, longname "vehicle info {}" },`
	`  type structure: {`
	`    paramOut { 20000, 20001, 20002, 20003, 20004, 20005, 20006, 20007,`
	`    20010, 20011, 20012, 20020, 20021, 461 }, convention mandatory`
	`} }`

A.6.8 array

The data type `array` of ASN.1 type `UNUM16` realized arrays of data types. The example in Table A.13 defines the data type 29 as array of the data type 30. The contained data type can be of any type. Recursions are not allowed, i.e. a data type cannot have itself as direct or indirect child.

Table A.13 — Array example

Example	`{ dataTypeId 29, type array: 30 },`

A.6.9 monitor

The data type `monitor` of ASN.1 type `SEQUENCE OF MonitorItem` realized Monitors. Table A.14 defines a monitor with two tests. Every test must have a `testId`. The parameters `decimalsPlaces`, `unit`, `factor`, `quotient` and `addend` are equal as defined in `numeric` (see A.6.2). The parameters min and max are not defined here but within the monitor value (see A.11.8).

Table A.14 — Monitor example

Example	{ dataTypeId 335, name { textId 10005, longname "oxygen sensor monitor" },
	  type monitor: {
	    { testId 1, decimalPlaces 3, unitId 17, factor 1, quotient 1000, addend 0 },
	    { testId 5, decimalPlaces 3, unitId 31, factor 1, quotient 1000, addend 0 }
	} }

A.6.10 octet

The data type `octet` of ASN.1 type `SNUM32` realized binary data as octet strings. The example in Table A.15 defines the data type 379 as octet of variable size and the data type 380 as an octet of 8 bytes.

Table A.15 — Octet example

Example	{ dataTypeId 379, name { textId 10094, longname "octetByte of variable size" },
	  type octet: 0 },
	{ dataTypeId 380, name { textId 10095, longname "octetByte with 8 byte" },
	  type octet: 8 }

A.7 DataParam

Table A.16 defines the data parameter attributes.

Table A.16 — DataParam attributes

Attributes	**Name**	**Description**	**Cvt**
	`rvId`	registered value identifier	M
	`name`	DisplayName of the data parameter (see A.17)	M
	`dataTypeId`	Reference to the data type identifier (see A.6.1)	M
	`accessType`	The access type of the data parameter is a bit string with a combination of following values: `r (0) = read, w (1) = write, x (2) = execute, i (3) internal, u (4) = user`	M
	`description`	Description of the data parameter	O
	`dataParamProperty`	The data parameter property is an enumeration with following values: `ecu-supported-info`, `sensor, actuator, ecu-internal-signal, ecu-internal-monitor, collection, control-function, fix` and `other`	M

Table A.16 *(continued)*

Example	{ rvId 461, name { textId 10511, shortname "VIN", longname "vehicle identification number" }, dataTypeId 360, accessType '10010'B, dataParamProperty ecu-internal-signal }, { rvId 1002, name { textId 10130, shortname "ECMB+", longname "engine control module voltage" }, dataTypeId 331, accessType '10000'B, dataParamProperty sensor }, { rvId 1123, name { textId 10131, shortname "HW_PART_NUMBER", longname "hardware part number" }, dataTypeId 334, accessType '10000'B, dataParamProperty ecu-internal-signal }, { rvId 2341, name { textId 10132, shortname "ECT", longname "engine coolant temperature" }, dataTypeId 333, accessType '10000'B, dataParamProperty sensor }, { rvId 7368, name { textId 10134, shortname "AIR_RDY", longname "secondary air system monitoring ready" }, dataTypeId 332, accessType '10000'B, dataParamProperty ecu-internal-monitor }, { rvId 20025, name { textId 10114, shortname "VehInfo", longname "vehicle info" }, dataTypeId 345, accessType '10000'B, dataParamProperty collection }

The data parameter is identified by the `rvId`, the registered value identifier. It is possible to have the same `rvId` on multiple ECUs to support, e.g. the battery voltage (ignition on) of every ECU. Further parameters are the unique `name` for the name of the parameter. The `dataTypeId` references the data type defined in A.6. The attributes `accessType` and `dataParamProperty` can be used for filtering the data parameters.

A.8 DataParamMapping

Table A.17 defines the mapping between DataParams and ECUs.

Table A.17 — DataParamMapping definition

Attributes	**Name**	**Description**	**Cvt**
	rvId	registered value identifier	M
	ecuId	ECU identifier	M
	arrayIndex	Index of an array for addressing, if the data parameter corresponding data type is an array	O
Example	{ rvId 1002, ecuId 17 }, { rvId 1123, ecuId 21 }		

A.9 DtcBase

Table A.18 defines the DtcBase.

Table A.18 — DtcBase definition

Attributes	Name	Description	Cvt
	`rDtcBaseId`	DTC base identifier	M
	`providerId`	Identifier of the provider of the DTC base, if the DTC base is provider specific	C1[a]
	`ecuId`	Identifier of the ECU supporting the DTC base, if the DTc base is only supported in this ECU	C2[a]
	`name`	DisplayName of the DTC base	M
	`description`	Description of the DTC base	O
	`dataParamList {}`	List of all related data parameter rvIds	C3[b]
	`dataParamMapping {}`	List of mappings between data parameter and ECU (see A.8)	C4[b]

Example	`{ rDtcBaseId 4, providerId 1,`
	`name { textId 20004, longname "Fuel Volume Regulator Control Circuit High" } },`
	`{ rDtcBaseId 5, ecuId 21,`
	`name { textId 20005, longname "Fuel Shutoff Valve 'A' Control Circuit/Open" }`
	`dataParamMapping {`
	`{ rvId 1002, ecuId 17 },`
	`{ rvId 2341, ecuId 17 }`
	`} }`
	`},`
	`{ rDtcBaseId 295, providerId 1,`
	`name { textId 20006, longname "Intake Air Temperature Too High" } },`
	`{ rDtcBaseId 49280, providerId 1,`
	`name { textId 20009, longname "Vehicle Communication Bus 'F'" } }`		

[a] Either C1 or C2 must be defined.

[b] Either C3 or C4 can be defined.

A.10 DtcSymptom

Table A.19 defines the DtcSymptom.

Table A.19 — DtcSymptom definition

Attributes	Name	Description	Cvt
	`rdtcSymptomId`	DTC symptom identifier	M
	`providerId`	Identifier of the provider of the DTC symptom, if the DTC symptom is provider specific	C1[a]
	`ecuId`	Identifier of the ECU supporting the DTC symptom, if the DTC symptom is only supported in this ECU	C2[a]
	`name`	DisplayName of the DTC symptom	M
	`description`	Description of the DTC symptom	O

Table A.19 *(continued)*

Example	{ rDtcSymptomId 4, providerId 1, name { textId 21004, longname "System Internal Failure" }, description { textId 22004, longname "This sub type is used for control module Internal Failures ..." } }, { rDtcSymptomId 8, providerId 1, name { textId 21008, longname "Bus Signal/Message Failure" } } }
[a] Either C1 or C2 must be defined.	

A.11 DataParamValue

A.11.1 DataType attributes

Table A.20 defines the DataParamValue attributes.

Table A.20 — DataParamValue attributes

Attributes	Name	Description	Cvt
	numeric	Numeric value as SNUM16	C0[c]
	lnumeric	Long numeric value as SNUM32	C1[c]
	string	String value as String	C2[c]
	displayName	DisplayName (see A.17)	C3[c]
	enumString	UNUM16 value, reference to the enumeration string item	C4[c]
	bitString	SNUM32 value containing the set bits of the bit string items	C5[c]
	structureMiss-ing	Dependence level of missing structure as numeric value as UNUM8	C6[c]
	array	Number of elements in the array as UNUM16	C7[c]
	monitor {}	List of MonitorValue elements with following attributes:	C8[c]
	— testValue	— SNUM32 testValue	M
	— testValueMin	— Minimal test value	O
	— testValueMax	— Maximal test value	O
	octet	Octet value	C9[c]
	error	Error id, if the parameter cannot be retrieved	C10[c]

[c] C0, C1, C2, C3, C4, C5, C6, C7, C8, and C9 are the choices of the data parameter value. Only one of C0 or C1 or C2 or C3 or C4 or C5 or C6 or C7 or C8 or C9 or C10 can be used.

The data parameter values must be mapped to the data type definition.

A.11.2 numeric, lnumeric

The data type value `numeric` is an `SNUM16`. The data type `lnumeric` is an `SNUM32`. So the value is very compact. The real value is calculated by Formula (1) with the attributes of the data type definition. To visualize the value, it must be rounded to the decimalPlaces defined in the data type. The example `numeric: 1250` corresponding to the data type `331` defined in A.6.2 must be interpreted as 12,50 V.

A.11.3 string

The data type value `string` is a `String`. The example `string: "VHJGH11763B65I860"` can be easily identified.

A.11.4 enumString

The data type value `enumString` is a `UNUM16` referencing the value of the corresponding data type `EnumStringItem`. The `enumString: 0` mapping to the `dataTypeId 332` defined in A.6.5 must be interpreted as '`no`'. An `enumString` mapping to another `dataTypeId` has a complete different meaning.

A.11.5 bitString

The data type value `bitString` is a `SNUM32`. It must be interpreted as bit mask as defined in Formula (2). The `bitString: 6` mapping to the `dataTypeId 33` defined in A.6.6 must be interpreted as 'snow and ice'. Of course, a `bitString` mapping to another `dataTypeId` has a complete different meaning.

A.11.6 structureMissing

The data type value `structureMissing` is a `UNUM8` containing the dependence level of the missing structure as numeric value. If a structure is expected in the data parameter value list, it contents are displayed. So only if the structure is optional, a `structureMissing` is used.

A.11.7 array

The data type value `array` is a `UNUM16` defining the size of the corresponding array. The data parameter value `array: 2` mapping to the `dataTypeId 29` defined in A.6.8 identifies an array with two elements with `dataTypeId 30`.

A.11.8 monitor

The data type value `monitor` is a `SEQUENCE OF MonitorValue`. It contains the test values and optional minimum and maximum of the MonitorItems of the corresponding data type definition. The example in Table A.21 mapping to the `dataTypeId 335` defined in A.6.9 provides test values `365` and `72` with min and max for `testId 1` and `5`.

Table A.21 — Monitor value example

Example	`value monitor: {`
	`{ testValue 365, testValueMin 365, testValueMax 365 },`
	`{ testValue 72, testValueMin 0, testValueMax 100 }`
	`}`

By using Formula (1) with the attributes of the data type definition, the example values must be interpreted as

```
testId1: testValue = 0.365 V (0.365 - 0.365 V)
testId5: testValue = 0.072 s (0.000 - 0.100 s)
```

A.11.9 octet

The data type value `octet` is a `OctetValue`. If the corresponding data type has an octet 0 (variable size), the length parameter defines the length of the octet string and data contains the data as octet string. If the corresponding data type has octet > 0 (fix size), the length parameter is not set. The example in Table A.22 defines an octet value of length 15.

Table A.22 — Octet value example

Example	`value octet: {`
	`    length 15, data '546869732069732064792064617461'H`
	`}`

A.11.10 error

The data type value `error` is a `SNUM32` defining the error retrieving the corresponding data parameter value. Every data parameter can result in an `error`. The error number references predefined errors.

A.12 VIErrorValue

Table A.23 defines the VIErrorValue.

Table A.23 — VIErrorValue definition

Attributes	**Name**	**Description**	**Cvt**
	`errorId`	error identifier	M
	`attribute {}`	List of data parameter values	O
Example	`{ errorId 1 },` `{ errorId 633, attribute { numeric: 32 } }`		

A.13 DataParamValueTS

Table A.24 defines the data parameter value time stamp (value with time stamp).

Table A.24 — DataParamValueTS definition

Attributes	**Name**	**Description**	**Cvt**
	`value`	data parameter value (see A.11)	M
	`timeInMillis`	Time in milliseconds since 1970	O

Example	`{ value numeric: 1250, time 0L },`
	`{ value enumString: 1, time 0L },`
	`{ value error: 422, time 0L }`		

A.14 DataParamValueMapping

Table A.25 defines the DataParamValueMapping.

Table A.25 — DataParamValueMapping definition

Attributes	**Name**	**Description**	**Cvt**
	`rvId`	registered value identifier	M
	`ecuId`	ECU identifier	O
	`value`	data parameter value (see A.11)	M
	`timeInMillis`	Time in milliseconds since 1970	O
Example	`{ rvId 1002, ecuId 17 },` `{ rvId 1123, ecuId 21 }`		