INTERNATIONAL STANDARD

ISO 13185-3

First edition
2018-06

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

Part 3: Unified vehicle interface protocol (UVIP) server and client API specification

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

Partie 3: Serveur du protocole unifié pour l'interface véhicule (UVIP) et spécification de l'API client

Reference number
ISO 13185-3:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

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

This document was prepared by Technical Committee ISO/TC 204, *Intelligent transport systems*.

Introduction

This document has been established to define the UGP client and server Java API of a common software interface to a vehicle UGP server to easily exchange vehicle information data amongst nomadic and/or mobile devices, cloud servers, vehicle servers 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.

A Nomadic Device (ND) becomes a Personal ITS station (P-ITS-S) if a Hardware Security Module (HSM) and software, that prohibits unauthorized access to an ITS-secured domain inside the ND, has been implemented.

This document defines the UGP client and server Java API protocol between the P-ITS-S and the UGP server in the vehicle.

The protocol specified in this document 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 document may be used by vehicle manufacturers to implement an interoperable UGP server in on-board communications modules that are allowed to interface with P-ITS-S(s). Through this interface, P-ITS-S(s) can access in-vehicle information provided to the UGP server. The means by which the UGP server obtains the in-vehicle information is outside the scope of this document.

The P-ITS-S applications need vehicle information data through an in-vehicle interface access method.

This document supports ITS applications based on a client-server model which allows clients on P-ITS-S to obtain data from ECUs in the in-vehicle networks (IVNs) through a common interface to a server located in a Vehicular ITS station (V-ITS-S) which in turn is acting as a gateway to the IVNs. The protocol implementation in the vehicle's UGP server may include the following features:

— the denial of access to the vehicle's UGP server data by unauthorized on-board and off-board test equipment;

— the denial of access to parts of the vehicle's UGP server data by unauthorized on-board and off-board test equipment (privacy);

— the identification of the vehicle's UGP server and the vehicle it is installed in;

— the list of in-vehicle connected ECUs to the vehicle's UGP server 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 3:
Unified vehicle interface protocol (UVIP) server and client API specification

1 Scope

This document specifies the server and client APIs of the Unified Gateway Protocol (UGP). Figure 1 shows an overview of the UGP client and server API. A UGP client application on a P-ITS-S communicates with a UGP server application on a V-ITS-S. The UGP client application implements the UGP client API using ISO 13185-2. The UGP server application implements the UGP server API using ISO 13185-2.

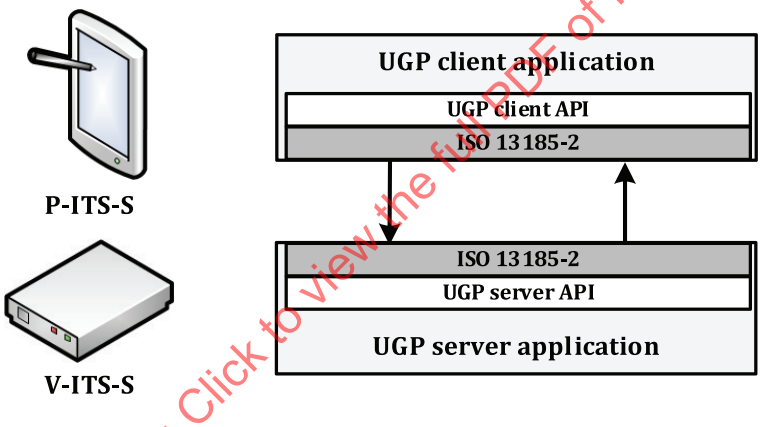

Figure 1 — UGP client and server API

NOTE This document does not define the UGP client and server API in other languages than Java.

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 (ITS) — Vehicle interface for provisioning and support of ITS services — Part 1: General information and use case definition*

ISO 13185-2, *Intelligent transport systems (ITS) — 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*

ISO 21217, *Intelligent transport systems — Communications access for land mobiles (CALM) — Architecture*

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, ISO 21217 and the following apply.

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

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at http://www.iso.org/obp

3.1
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 it's packaging and labelling claims to be.

3.2
authorisation
prescription that a particular behaviour shall not be prevented

[SOURCE: ISO 17419, 3.1]

3.3
nomadic device
ND
provides communications connectivity via equipment such as cellular telephones, mobile wireless broadband (WIMAX, HC-SDMA, etc.), WiFi, etc. and includes short range links, such as Bluetooth, Zigbee

Note 1 to entry: Nomadic devices do not necessarily implement ITS-specified security, e.g. hardware security module.

3.4
privacy
choice made by the vehicle owner to grant information access for a special tool or user, or if the data should be used in the vehicle/off-board systems or not

Note 1 to entry: The privacy/authorization information is kept as master information off-board and synchronised to the on-board V-ITS-S.

3.5
UGP client
entity that implements the UGP client services

EXAMPLE ND, P-ITS-S.

3.6
UGP server
entity that implements the UGP server services

EXAMPLE V-ITS-S.

3.7
unified gateway protocol
UGP
application layer protocol to enable UGP clients to access data from the UGP server

4 Abbreviated terms

API	application programming interface
ASN.1	abstract syntax notation one
A_PDU	application layer protocol data unit
A_SDU	application layer service data unit
Conf	confirmation
CRC	cyclic redundancy check
DTC	diagnostic trouble code
ECU	electronic control unit
GUI	graphical user interface
Ind	indication
ITS	intelligent transport systems
ND	nomadic device
OSI	open systems interconnection
P-ITS-S	personal – intelligent transport system – station
Req	request
Resp	response
UGP	unified gateway protocol
V-ITS-S	vehicle – intelligent transport system – station

5 Conventions

This document is based on the conventions discussed in the OSI Service Conventions (ISO/IEC 10731:1994)[1] 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 may convey parameters.

Figure 2 shows a high level flow diagram of UGP.

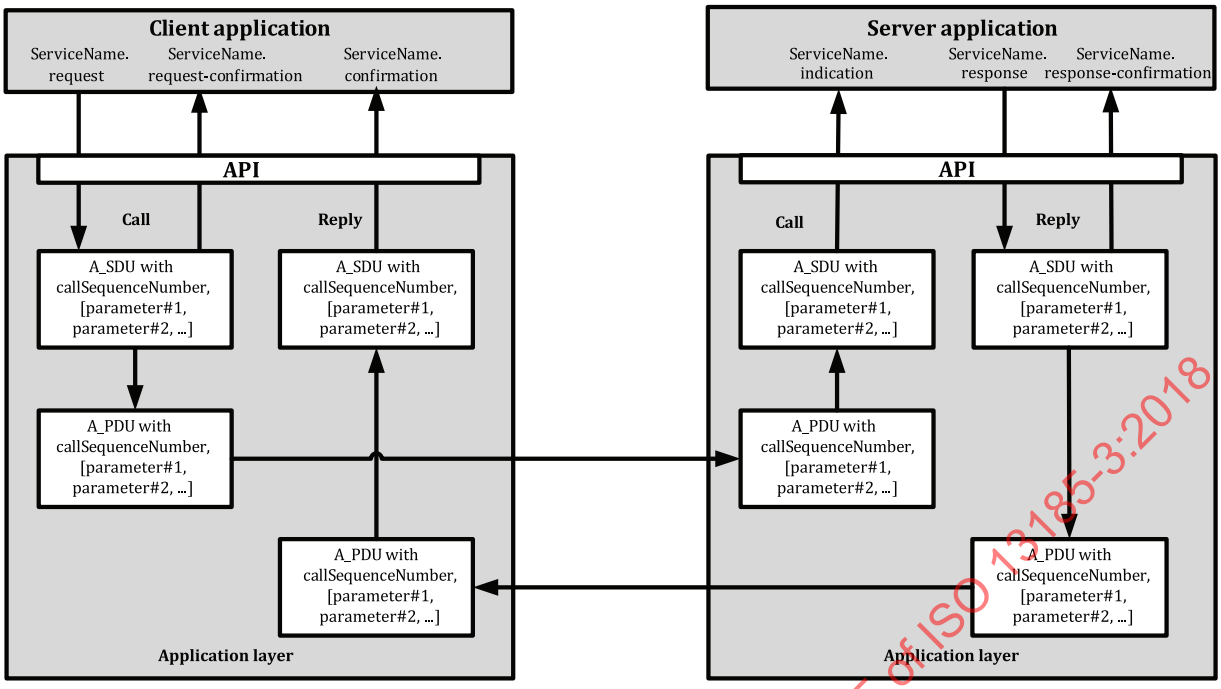

Figure 2 — UGP client-server relationship

This document defines services using the six service primitives: request, req_confirm, indication, response, rsp_confirm and confirmation as defined in ISO 13185-2:2015, Clause 6.

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 document, only the request and response service primitives are listed.

6 UGP API concept

6.1 UGP client and server architecture

Figure 3 shows the UGP client and server architecture. On the P-ITS-S there is a UGP client application and on the V-ITS-S there is a UGP server application.

The UGP client application uses the interface IUGPRequest to request services and the interface IUGPConfirmation to receive the corresponding confirmation. In the confirmation implementation another request can be called or the GUI can be updated. The UGPRequest Implementation shall encode the request into a corresponding ISO 13185-2 Call. Before calling the IUGPConfirmation the UGP client application shall decode and interpret the ISO 13185-2 Reply.

The UGP server application shall decode ISO 13185-2 Calls and interpret it to call its interface IUGPIndication. The corresponding implementation shall get data, DTCs, etc. and to create the corresponding response by calling the interface IUGPResponse. The UGPResponseImplementation shall encode the response into an ISO 13185-2 Reply.

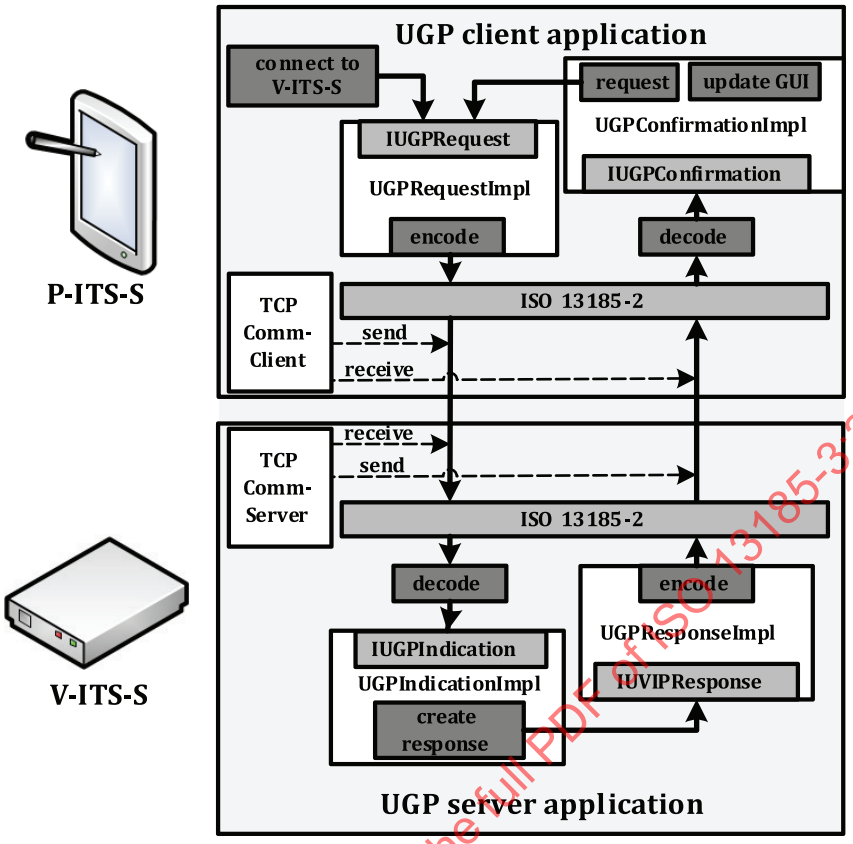

Figure 3 — UGP client and server architecture

6.2 UGP communication flow

Figure 4 shows the UGP communication flow. The UGP client application shall create a connection to the V-ITS-Ss UGP server application. Then it calls an authenticationRequest which is encoded into an ISO 13185-2 authenticationCall. The UGP server application decodes the authenticationCall and calls the authenticationIndication. If the authenticationKey is not OK for the V-ITS-S, the UGPIndication implementation creates a negativeResponse which will be decoded as ISO 13185-2 globalNegativeReply. The UGP client application decodes the globalNegativeReply and calls a negativeConfirmation. Usually the UGP client application should update its GUI with an error message “Not allowed to access”.

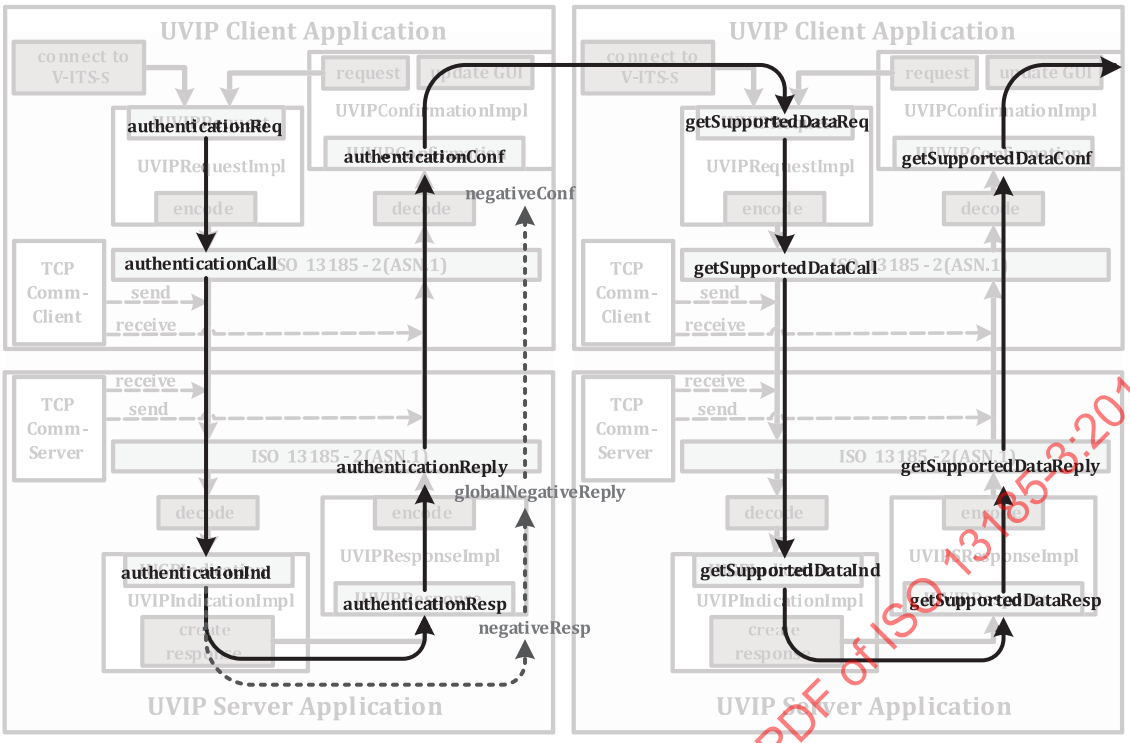

Figure 4 — UGP communication flow

If the authenticationKey is OK for the V-ITS-S, the UGPIndication implementation calls an authenticationResponse which will be decoded as ISO 13185-2 authenticationReply. The UGP client application decodes the authenticationReply and calls an authenticationConfirmation. The authenticationConfirmation can indicate the authentication authorization bits but shall call now a getSupportedDataRequest.

The UGP client application encodes the getSupportedDataRequest into an ISO 13185-2 getSupportedDataCall. The UGP server application receives and decodes it and calls a getSupportedDataIndication which creates the requested supported data infos in a getSupportedDataResponse. This is decoded into an ISO 13185-2 getSupportedDataReply. The UGP client application receives and decodes the getSupportedDataReply and calls a getSupportedDataConfirmation.

The data parameters or DTCs can be requested by getValueReq or getDtcInfoReq, etc.

6.3 UGP interfaces and service primitives

Table 1 defines the UGP interfaces and service primitives.

Table 1 — UGP interfaces and service primitives

IUGPRequest	IUGPIndication	IUGPResponse	IUGPConfirmation
authenticationReq	authenticationInd	authenticationResp	authenticationConf
getSupportedDataReq	getSupportedDataInd	getSupportedDataResp	getSupportedDataConf
getValueReq	getValueInd	getValueResp	getValueConf
setValueReq	setValueInd	positiveResp	positiveConf
controlValueReq	controlValueInd	controlValueResp	controlValueConf
getDtcInfoReq	getDtcInfoInd	getDtcInfoResp	getDtcInfoConf
clearDtcInfoReq	clearDtcInfoInd	positiveResp	positiveConf

Table 1 *(continued)*

IUGPRequest	IUGPIndication	IUGPResponse	IUGPConfirmation
enablePassThruReq	enablePassThruInd	positiveResp	positiveConf
listFileReq	listFileInd	listFileResp	listFileConf
manageFileUploadReq	manageFileInd	manageFileResp	manageFileConf
manageFileDownloadReq	manageFileInd	manageFileResp	manageFileConf
manageFileDeleteReq	manageFileInd	manageFileResp	manageFileConf
resetReq	resetInd	positiveResp	positiveConf
stopServiceReq	stopServiceInd	positiveResp	positiveConf

7 UGP Java client API

7.1 UGP Java client API requests

7.1.1 Overview

Corresponding to the UGP services defined in ISO 13185-2, the UGP Java client requests API (IUGPRequest) contains request services that can be called by any application to execute a UGP service (Req). The UGP Java client API confirmations (see 7.2) handle the corresponding replies.

Table 2 defines the UGP Java client API requests (IUGPRequest).

Table 2 — UGP Java client API requests (IUGPRequest)

Service	API definition
authenticationReq	UGPPackage authenticationReq(String authenticationKey) throws VIException;
getSupportedDataReq	UGPPackage getSupportedDataReq(SupportedDataFilter supportedDataFilter) throws VIException; UGPPackage getSupportedDataReq(SupportedDataFilter supportedDataFilter, Integer[] ecuIds, AccessType accessType, Integer dataParamProperty) throws VIException;
getValueReq	UGPPackage getValueReq(int testInterval, Integer rvId, ComplexCondition condition) throws VIException; UGPPackage getValueReq(int testInterval, Integer[] rvIds, ComplexCondition condition) throws VIException; UGPPackage getValueReq(int testInterval, Vector<DataParamMapping> mappings, ComplexCondition condition) throws VIException;
setValueReq	UGPPackage setValueReq(Vector<DataParamValueMapping> valueMappings) throws VIException;
controlValueReq	UGPPackage controlValueReq(int testInterval, Integer rvId, Vector<DataParamValue> values, ExecutionType execute) throws VIException; UGPPackage controlValueReq(int testInterval, Integer[] rvIds, Vector<DataParamValue> values, ExecutionType execute) throws VIException;
getDtcInfoReq	UGPPackage getDtcInfoReq(int testInterval, Integer ecuId, boolean withEnvData, ComplexCondition condition) throws VIException; UGPPackage getDtcInfoReq(int testInterval, Integer[] ecuIds, boolean withEnvData, ComplexCondition condition) throws VIException;
clearDtcInfoReq	UGPPackage clearDtcInfoReq(Integer ecuId) throws VIException; UGPPackage clearDtcInfoReq(Integer[] ecuIds) throws VIException;
enablePassThruReq	UGPPackage enablePassThruReq(String label, String key) throws VIException;

Table 2 *(continued)*

Service	API definition
listFileReq	UGPPackage listFileReq(FileType fileType) throws VIException;
manageFileUploadReq	UGPPackage manageFileUploadReq(FileType fileType, File f) throws VIException; UGPPackage manageFileDownloadReq(FileType fileType, String filename) throws VIException; UGPPackage manageFileDeleteReq(FileType fileType, String filename) throws VIException;
resetReq	UGPPackage resetReq(ResetType resetType) throws VIException;
stopServiceReq	UGPPackage stopServiceReq(int callSequenceNumber) throws VIException;

All request services return a UGPPackage containing the ASN.1 message to send to the UGP server or throw a VIException if one of the parameters is invalid.

7.1.2 authenticationReq

Table 3 defines the authentication request service.

Table 3 — Definition of the authentication request service

SP	**authenticationReq**	Initiates the communication to the V-ITS-S by requesting the users V-ITS-S authentication key.	
Conf	**positiveConf**		
Param	**Name**	**Type**	**Description**
	`authenticationKey`	String	Public key for access to a V-ITS-S
API	`UGPPackage authenticationReq(String authenticationKey) throws VIException;`		
e.g.	`UGPClient.authenticationReq("0vrmYdc6ziscBdMhGphiT6m0a0D2dNLCnETRSlg");`		

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

7.1.3 getSupportedDataReq

Table 4 defines the getSupportedData request service.

Table 4 — Definition of the getSupportedData request service

SP	**getSupportedDataReq**	Request the supported data parameters from the V-ITS-S.	
Conf	**getSupportedDataConf**		
Param	**Name**	**Type**	**Description**
	`supportedDataFilter`	`SupportedDataFilter (Enumeration)`	Filters the supported data to: `vehicle-info-only, with-ecu-data, without-ecu-data`
	`ecuIds`	`Integer[]`	List of the ECU identifer of the in-vehicle network ECUs from where the data parameters should be retrieved. If ecuList is empty or null the data parameters are not filtered by ECU.
	`accessType`	`AccessType`	Filters the supported data parameters to the specified access type. If no `accessType` is defined the access types are not filtered.
	`dataParamProperty`	`Integer`	Filters the supported data parameters to the specified data parameter property. If no `dataParamProperty` is defined, the data parameter properties are not filtered.

API	`UGPPackage getSupportedDataReq(SupportedDataFilter supportedDataFilter, Integer[] ecuIds, AccessType accessType, Integer dataParamProperty) throws VIException;`
	
	`enum SupportedDataFilter { vehicle_info_only(0), with_ecu_data(1), without_ecu_data(2); }`
	`/** predefined AccessTypes **/`
	`static AccessType read_only, write_only, read_write, execute, write_internal, user_input;`		

e.g.	`UGPClient.getSupportedDataReq(SupportedDataFilter.without-ecu-data, null, null, null);` `UGPClient.getSupportedDataReq(SupportedDataFilter.with-ecu-data, new int[] { 17 }, AccessType.read-only, null);`		

ASN.1	`GetSupportedDataCall:: = SEQUENCE {`
	`supportedDataFilter SupportedDataFilter DEFAULT with-ecu-data,`
	`ecuList SEQUENCE OF Identifier OPTIONAL,`
	`accessType AccessType OPTIONAL,`
	`dataParamProperty DataParamProperty OPTIONAL,`
	`...`
	`}`		

7.1.4 getValueReq

Table 5 defines the getValue request service.

Table 5 — Definition of the getValue request service

SP	**getValueReq**	Request the data parameter values from the V-ITS-S.	
Conf	**getValueConf**		
Param	**Name**	**Type**	**Description**
	`testInterval`	`int`	If testInterval is 0 the result should be sent only once if the condition is true or not set. If testInterval > 0 it contains a number of milliseconds and each time the testInterval has expired and the condition is true or not set the getValueConf will be sent.
	`rvId`	`Integer`	RvId of a single data parameter to retrieve.
	`rvIds`	`Integer[]`	List of rvIds of the data parameters to retrieve.
	`mappings`	`Vector<DataParamMapping>`	List of mappings between ecuId and rvId of the corresponding data parameters.
	`condition`	`ComplexCondition`	Condition that will be checked once (if testInterval = 0) or every testInterval milliseconds (testInterval > 0).

API	`UGPPackage getValueReq(int testInterval, Integer rvId,`
	`  ComplexCondition condition) throws VIException;`
	
	`UGPPackage getValueReq(int testInterval, Integer[] rvIds,`
	`  ComplexCondition condition) throws VIException;`
	
	`UGPPackage getValueReq(int testInterval, Vector<DataParamMapping> mappings,`
	`  ComplexCondition condition) throws VIException;`		

e.g.	`UGPClient.getValueReq(0, 2344, null);`
	`UGPClient.getValueReq(0, 20025, null);`
	
	`UGPClient.getValueReq(0, new Integer[] { 1002, 7368, 2341, 1123 }, null);`
	
	`Vector<DataParamMapping> mappings = new Vector<DataParamMapping>;`
	`UGPClient.addDataParamMapping(mappings, 17, 1002);`
	`UGPClient.addDataParamMapping(mappings, 17, 7368);`
	`UGPClient.addDataParamMapping(mappings, 51, 2341);`
	`ComplexCondition conditionECTgt110 = UGPClient.createDataParamCondition(51, 2341,`
	`  OperatorType.gt, DataParamCondValue.numeric(1100));`
	`UGPClient.getValueReq(5000, mappings, conditionECTgt180);`		

ASN.1	`GetValueCall:: = SEQUENCE {`
	`  testInterval  SNUM32,`
	`  dataParamList  SEQUENCE OF Identifier  OPTIONAL,`
	`  dataParamMapping  SEQUENCE OF DataParamMapping  OPTIONAL,`
	`  condition  ComplexCondition  OPTIONAL,`
	`  ...`
	`}`		

7.1.5 setValueReq

Table 6 defines the setValue request service.

Table 6 — Definition of the setValue request service

SP	setValueReq	Request the data parameter values from the V-ITS-S.	
Conf	positiveConf		
Parameter	Name	Type	Description
	valueMapping	Vector <DataParamValueMapping>	List of data parameter value mappings
API	setValueReq(Vector<DataParamValueMapping> valueMappings) throws VIException;		
e.g.	Vector<DataParamValueMapping> valueMappings = new Vector<DataParamValueMapping>(); UGPClient.addDataParamValueMapping(17, 7368, DataParamValue.enumString(0)); UGPClient.setValueReq(valueMappings);		

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

7.1.6 controlValueReq

Table 7 defines the controlValue request service.

Table 7 — Definition of the controlValue request service

SP	controlValueReq	Request the data parameter values from the V-ITS-S.	
Conf	controlValueConf		
Parameter	Name	Type	Description
	testInterval	int	If testInterval is 0 the confirmation is sent only once when the control function completes. If testInterval > 0 the confirmation is sent each time the testInterval in milliseconds has expired.
	rvId	Integer	RvId of a single control-function.
	rvIds	Integer[]	List of rvIds of the control-functions.
	mappings	Vector <DataParamMapping>	List of mappings between ECU and control-function to start or stop.
	values	Vector <DataParamValue>	List of data parameter values as parameters for the control function.
	execute	ExecutionType	Type of the execution with values start and stop to start or stop the control function.

Table 7 *(continued)*

SP	**controlValueReq**	Request the data parameter values from the V-ITS-S.
Conf	**controlVal-ueConf**	
API	UGPPackage controlValueReq(int testInterval, Integer rvId, Vector<DataParamValue> values, ExecutionType execute) throws VIException; UGPPackage controlValueReq(int testInterval, Integer[] rvIds, Vector<DataParamValue> values, ExecutionType execute) throws VIException; UGPPackage controlValueReq(int testInterval, Vector<DataParamMapping> v, Vector<DataParamValue> values, ExecutionType execute) throws VIException;	

e.g.	Vector<DataParamMapping> mappings = new Vector<DataParamMapping>();
	UGPClient.addDataParamMapping(mappings, 22, 0xfa002f00l);
	Vector<DataParamValue> values = new Vector<DataParamValue>();
	values.addElement(DataParamValue.lnumeric(0xfd009b00l));
	values.addElement(DataParamValue.enumString(3));
	values.addElement(DataParamValue.numeric(60));
	UGPClient.controlValueReq(0, mappings, values, ExecutionType.start);
	
	values = new Vector<DataParamValue>();
	values.addElement(DataParamValue.lnumeric(0xfd0001551));
	values.addElement(DataParamValue.enumString(3));
	values.addElement(DataParamValue.numeric(7));
	values.addElement(DataParamValue.error(0)); //4 times ...
	UGPClient.controlValueReq(0, mappings, values, ExecutionType.start);	

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, ...
	}	

7.1.7 getDtcInfoReq

Table 8 defines the getDtcInfo request service.

Table 8 — Definition of the getDtcInfo request service

SP	getDtcInfoReq	Request the current DTCs.	
Conf	getDtcInfoConf		
Param	Name	Type	Description
	testInterval	int	If testInterval is 0 the result should be sent only once if the condition is true or not set. If testInterval > 0 it contains a number of milliseconds and each time the testInterval has expired and the condition is true or is not set the getDtcInfoConf will be sent.
	ecuId	Integer	Filters the DTCs to the ECU defined by ecuId. If ecuId is null, the DTCs of all ECUs are retrieved.
	ecuIds	Integer[]	Filters the DTCs to the ECus defined by the ecuIds. If ecuIds are null, the DTCs of all ECUs are retrieved.
	withEnvData	boolean	false: retrieve no environment data true: retrieve configured environment data
	condition	ComplexCondition	Condition that will be checked once (if testInterval = 0) or every testInterval milliseconds (testInterval > 0).
API	UGPPackage getDtcInfoReq(int testInterval, int ecuId, boolean withEnvData, ComplexCondition condition) throws VIException; UGPPackage getDtcInfoReq(int testInterval, int[] ecuIds, boolean withEnvData, ComplexCondition condition) throws VIException;		
e.g.	UGPClient.getDtcInfoReq(0, 32, false, null); UGPClient.getDtcInfoReq(10000, null, true, null);		

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,
	...
	}		

7.1.8 clearDtcInfoReq

Table 9 defines the clearDtcInfo request service.

Table 9 — Definition of the clearDtcInfo request service

SP	clearDtcInfoReq	Request to clear the DTCs of the defined ECUs.	
Conf	positiveConf		
Parameter	Name	Type	Description
	ecuId	Integer	Clears the DTCs of the defined ECU. If ecuId is null, the DTCs of all ECUs are cleared.
	ecuIds	Integer[]	Clears the DTCs of the defined ECUs. If ecuIds are null, the DTCs of all ECUs are cleared.

Table 9 *(continued)*

SP	**clearDtcInfoReq**	Request to clear the DTCs of the defined ECUs.
Conf	**positiveConf**	

API	`UGPPackage clearDtcInfoReq(Integer ecuId) throws VIException;`
	
	`UGPPackage clearDtcInfoReq(Integer[] ecuIds) throws VIException;`	

e.g.	`UGPClient.clearDtcInfoReq(null);`
	
	`UGPClient.clearDtcInfoReq(206);`	

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

7.1.9 enablePassThruReq

Table 10 defines the enablePassThru request service.

Table 10 — Definition of the enablePassThru request service

SP	**enablePassThruReq**	Request to enable or disable the pass thru modus.	
Conf	**positiveConf**		
Parameter	**Name**	**Type**	**Description**
	`label`	`String`	Label of the pass thru seed to enable or disable the pass thru
	`key`	`String`	Key to enable the pass thru; if the key is null, the pass thru will be disabled
API	`UGPPackage enablePassThruReq(String label, String key) throws VIException;`		

e.g.	`UGPClient.enablePassThruReq("ENH_DIAG_PASS_THRU_SEED", "flkja0932jdla9323jddff3d");`
	
	`UGPClient.enablePassThruReq("ECU_PRG_PASS_THRU_SEED", null);`		

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

7.1.10 listFileReq

Table 11 defines the listFile request service.

Table 11 — Definition of the listFile request service

SP	listFileReq	Request a list of files of the given fileType from the V-ITS-S.	
Conf	listFileConf		
Parameter	Name	Type	Description
	fileType	FileType	Type of the file: core, configuration, log, snapshot, calibration.

API	UGPPackage listFileReq(FileType fileType) throws VIException;
	
	enum FileType {
	core(0), configuration(1), log(2), snapshot(3), calibration(4);
	}		

e.g.	UGPClient.listFileReq(FileType.configuration);		

ASN.1	ListFileCall:: = SEQUENCE {
	fileType FileType,
	...
	}
	FileType:: = ENUMERATED { core, configuration, log, snapshot, calibration, ...
	}		

7.1.11 manageFileUploadReq

Table 12 defines the manageFileUpload request service.

Table 12 — Definition of the manageFileUpload request service

SP	manageFileUploadReq	Request to manage a file.	
Conf	manageFileConf		
Parameter	Name	Type	Description
	fileType	FileType	Type of the file: core, configuration, log, snapshot, calibration.
	f	File	The file to upload.

Table 12 *(continued)*

SP	**manageFileUp-loadReq**	Request to manage a file.
Conf	**manageFileConf**	

API	`UGPPackage manageFileUploadReq(FileType type, File f) throws VIException;`
	
	`enum FileType {`
	`core(0), configuration(1), log(2), snapshot(3), calibration(4);`
	`}`	

e.g.	`File f = new File(DIR, "passcar_hyundai_i30_2007_2007110402");` `UGPClient.manageFileUploadReq(FileType.configuration, f);`	

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

7.1.12 manageFileDownloadReq

Table 13 defines the manageFileDownload request service.

Table 13 — Definition of the manageFileDownload request service

SP	**manageFileDown-loadReq**	Request to manage a file.	
Conf	**manageFileConf**		
Pa-ram-eter	**Name**	**Type**	**Description**
	`fileType`	`FileType`	Type of the file: `core, configuration, log, snapshot, calibration`.
	`filename`	`String`	The filename to download or delete.
API	`UGPPackage manageFileDownloadReq(FileType type, String filename) throws VIException;`		
e.g.	`UGPClient.manageFileDownloadReq(FileType.configuration,` `"passcar_hyundai_i30_2007_2007110402");`		

7.1.13 manageFileDeleteReq

Table 14 defines the manageFileDelete request service.

Table 14 — Definition of the manageFileDelete request service

SP	**manageFileReq**	Request to manage a file.	
Conf	**manageFileConf**		
Parameter	**Name**	**Type**	**Description**
	`fileType`	`FileType`	Type of the file: `core, configuration, log, snapshot, calibration`.
	`filename`	`String`	The filename to download or delete.
API	`UGPPackage manageFileDeleteReq(FileType type, String filename) throws VIException;`		
e.g.	`UGPClient.manageFileDeleteReq(FileType.configuration, "passcar_hyundai_i30_2007_2007110402");`		

7.1.14 resetReq

Table 15 defines the reset request service.

Table 15 — Definition of the reset request service

SP	**resetReq**	Request a reset.	
Conf	**positiveConf**		
Parameter	**Name**	**Type**	**Description**
	`resetType`	`FileType`	Type of the reset: `reboot, shutdown, restart`

API	`UGPPackage resetReq(ResetType resetType) throws VIException;`
	
	`enum ResetType {`
	`  reboot(0), shutdown(1), restart(2);`
	`}`		

e.g.	`UGPClient.resetReq(ResetType.reboot);`		

ASN.1	`ResetCall:: = SEQUENCE {`
	`  resetType  ResetType,`
	`  ...`
	`}`
	`ResetType:: = ENUMERATED { reboot, shutdown, restart, ...`
	`}`		

7.1.15 stopServiceReq

Table 16 defines the stopService request service.

Table 16 — Definition of the stopService request service

SP	**stopServiceReq**	Request a service termination.	
Conf	**positiveConf**		
Parameter	**Name**	**Type**	**Description**
	`callSequenceNumber`	`int`	The sequence number of the asynchronous service to stop.
API	`UGPPackage stopServiceReq(int callSequenceNumber) throws VIException;`		
e.g.	`UGPPackage UGPPackage = UGPClient.getValueReq(1000, 2344, null);` `UGPClient.stopServiceReq(UGPPackage.getCallSequenceNumber());`		

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

7.2 UGP Java Client API confirmations

7.2.1 Overview

Table 17 defines the UGP Java Client API confirmations (IUGPConfirmation) that shall be implemented to handle the confirmations received from the UGP Server. In a confirmation service the results are interpreted and often another request service is called and sent.

Table 17 — UGP Java Client API confirmations (IUGPConfirmation)

Service	API definition
authenticationConf	void authenticationConf(AuthorizationBits authorization) throws NumberedException;
getSupportedDataConf	void getSupportedDataConf(Vector<EcuDataParam> params) throws NumberedException;
getValueConf	void getValueConf(Vector<EcuDataParam> params) throws NumberedException;
controlValueConf	void controlValueConf(ControlValueReply reply) throws NumberedException;
getDtcInfoConf	void getDtcInfoConf(Vector<DtcInfo> dtcInfos) throws NumberedException;
listFileConf	void listFileConf(Vector<FileInfo> fileInfos) throws NumberedException;
manageFileConf	void manageFileConf() throws NumberedException;
positiveConf	void positiveConf(int callChoice, int callSequenceNumber);
negativeConf	void negativeConf(final NumberedException e);

All confirmation services throw a NumberedException which generalizes VIException and CommException. A VIException indicates a problem while creating a new request service. A CommException indicates a problem while sending this request.

7.2.2 authenticationConf

Table 18 defines the authentication confirmation service.

Table 18 — Definition of the authentication confirmation service

SP	**authenticationConf**	Replies the authenticationReq by returning a bit mask for the user's authorization.	
Req	**authenticationReq**		
Parameter	**Name**	**Type**	**Description**
	authorization	AuthorizationBits	Bit mask with all authorisations.

API	void authenticationConf(AuthorizationBits authorization) throws NumberedException;
	
	class AuthorizationBits extends BitString {
	public static final int get_value_extended_access = 0;
	public static final int set_value_access = 1;
	public static final int control_value_access = 2;
	public static final int enable_pass_thru_access = 3;
	public static final int file_download_access = 4;
	public static final int file_upload_access = 5;
	public static final int file_delete_access = 6;
	...
	}		

e.g.	sendCall(getSupportedDataReq(SupportedDataFilter.with_ecu_data, null, null, null));		

ASN.1	AuthenticationReply:: = SEQUENCE {
	authorization AuthorizationBits,
	...
	}		

7.2.3 getSupportedDataConf

Table 19 defines the getSupportedData confirmation service.

Table 19 — Definition of the getSupportedData confirmation service

SP	getSupportedDataConf	Replies the getSupportedDataReq by returning the data types, data parameters and their mappings to the ECUs.	
Req	getSupportedDataReq		
Parameter	Name	Type	Description
	ecuDataParams	Vector <EcuDataParam>	List of the requested supported ECU Data Parameters containing at least an id (combined from ecuId and rvId) and references to its ECU, Data Parameter, Data Type, Unit, its value and the values timestamp.

API	void getSupportedDataConf(Vector<EcuDataParam> params) throws NumberedException;
	
	public class EcuDataParam {
	long id; //highword: ecuId; lowword: rvId
	Ecu ecu;
	DataParam param;
	DataType type;
	Unit unit;
	DataParamValue value;
	Long timeInMillis; /* OPTIONAL */
	...
	}		

e.g.	updateGUI(params); sendCall(getValueReq(testInterval, params, null));		

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

7.2.4 getValueConf

Table 20 defines the getValue confirmation service.

Table 20 — Definition of the getValue confirmation service

SP	getValueConf	Replies the getValueReq by returning the data parameters with its values.	
Req	getValueReq		
Parameter	Name	Type	Description
	ecuDataParams	Vector <EcuDataParam>	List of requested ECU data parameters containing its values.
API	void getValueConf(Vector<EcuDataParam> params) throws NumberedException;		
e.g.	updateGUI(params);		

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

7.2.5 controlValueConf

Table 21 defines the controlValue confirmation service.

Table 21 — Definition of the controlValue confirmation service

SP	**controlValueConf**	Replies the controlValueReq by returning the control function`s execution status and outgoing parameters with values.	
Req	**controlValueReq**		
Parameter	**Name**	**Type**	**Description**
	`status`	`ExecutionStatus`	Execution status of the control function with the values: `in-progress, pass, fail`.
	`ecuDataParams`	`Vector <EcuDataParam>`	List of requested outgoing ECU data parameters containing its values.

API	`void controlValueConf(ExecutionStatus status, Vector<EcuDataParam> params) throws NumberedException;`
	
	`enum ExecutionStatus { in_progress(0), pass(1), fail(2); }`		

e.g.	`updateGUI(params);`		
ASN.1	`ControlValueReply:: = SEQUENCE { status ExecutionStatus, value SEQUENCE OF DataParamValue, ... }`		

7.2.6 getDtcInfoConf

Table 22 defines the getDtcInfo confirmation service.

Table 22 — Definition of the getDtcInfo confirmation service

SP	getDtcInfoConf	Replies the getDtcInfoReq by returning DtcInfos.	
Req	getDtcInfoReq		
Parameter	Name	Type	Description
	dtcInfos	Vector<DtcInfo>	List of requested DTC infos containing at least an id (combined from rDtcBaseId and rDtcSymptomId) and references to its DtcBase, DtcSymptom, ECU. The complementary contains infos about status, severity and class. If in the corresponding request withEnvData was true, the DtcInfo addtionally contains the environment data as list of EcuDataParams.

API	`void dtcInfoConf(Vector<DtcInfo> dtcInfos) throws NumberedException;`
	
	`public class DtcInfo {`
	`  int id; //highword = rDtcBaseId; lowword = rDtcSymptomId`
	`  DtcBase dtcBase;`
	`  DtcSymptom dtcSymptom;`
	`  Ecu ecu;`
	`  DtcComplementary complementary; /* BIT STRING */`
	`  Vector<EcuDataParam> envData;`
	`}`		

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

7.2.7 listFileConf

Table 23 defines the listFile confirmation service.

Table 23 — Definition of the listFile confirmation service

SP	listFileConf	Replies the listFileReq by returning a list of FileInfos.	
Req	listFileReq		
Parameter	Name	Type	Description
	fileInfos	Vector <FileInfo>	List of requested file information containing at least the file name. Additionally it can contain a version.

API	void listFileConf(Vector<FileInfo> fileInfos) throws NumberedException;
	
	public class FileInfo {
	String fileName;
	String version; /* OPTIONAL */
	}		

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

7.2.8 manageFileConf

Table 24 defines the manageFile confirmation service.

Table 24 — Definition of the manageFile confirmation service

SP	manageFileConf	Replies the manageFileUploadReq, manageFileDownloadReq or manageFileDeleteReq.	
Req	manageFileUploadReq manageFileDownloadReq manageFileDeleteReq		
Parameter	Name	Type	Description
	---	---	To confirm manageFileUploadReq and manageFileDeleteReq no parameters are needed.
API	void manageFileConf() throws NumberedException;		

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

7.2.9 positiveConf

Table 25 defines the positive confirmation service.

Table 25 — Definition of the positive confirmation service

SP	positiveConf	Replies all requests without specific confirmation.	
Parameter	Name	Type	Description
	choice	int	Enumeration of service choice: `setValueCall`, `clearDtcInfoCall`, `enablePassThruCall`, `keyOffOnResetCall`, `stopServiceCall`
API	`void positiveConf(int callChoice, int callSequenceNumber);`		

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

7.2.10 negativeConf

Table 26 defines the negative confirmation service.

Table 26 — Definition of the negative confirmation service

SP	negativeConf	Replies all requests if a failure occurs on the UGP Server side.	
Parameter	Name	Type	Description
	e	NumberedException	The exception occurred on the UGP server side.
API	`void negativeConf(final NumberedException e);`		

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

8 UGP Java server API

8.1 UGP Java server API indications

8.1.1 Overview

The UGP Java server API contains indication services that shall be implemented to reply to. At the execution end of an indication service the corresponding UGP Java server API response should be created (see 8.2).

Table 27 defines the UGP Java server API indications (IUGPIndication).

Table 27 — UGP Java server API indications (IUGPIndication)

Service	API definition
authenticationInd	UGPPackage authenticationInd(int callSequenceNumber, AuthenticationCall call) throws VIException;
getSupportedDataInd	UGPPackage getSupportedDataInd(int callSequenceNumber, GetSupportedDataCall call) throws VIException;
getValueInd	UGPPackage getValueInd(int callSequenceNumber, GetValueCall call) throws VIException;
setValueInd	UGPPackage setValueInd(int callSequenceNumber, SetValueCall call) throws VIException;

Table 27 *(continued)*

Service	API definition
controlValueInd	UGPPackage controlValueInd(int callSequenceNumber, ControlValueCall call) throws VIException;
getDtcInfoInd	UGPPackage getDtcInfoInd(int callSequenceNumber, GetDtcInfoCall call) throws VIException;
clearDtcInfoInd	UGPPackage clearDtcInfoInd(int callSequenceNumber, ClearDtcInfoCall call) throws VIException;
enablePassThruInd	UGPPackage enablePassThroughInd(int callSequenceNumber, EnablePassThruCall call) throws VIException;
listFileInd	UGPPackage listFileInd(int callSequenceNumber, ListFileCall call) throws VIException;
manageFileInd	UGPPackage manageFileInd(int callSequenceNumber, ManageFileCall call) throws VIException;
resetInd	UGPPackage resetInd(int callSequenceNumber, ResetCall call) throws VIException;
stopServiceInd	UGPPackage stopServiceInd(int callSequenceNumber, int stopSequenceNumber) throws VIException;

All indicationServices include the callSequenceNumber of the call as first parameter, throw a VIException if the service fails, and return an UGPPackage containing the positive response. The VIException will be converted into a negativeResponse by the surrounding framework.

8.1.2 authenticationInd

Table 28 defines the authentication indication service.

Table 28 — Definition of the authentication indication service

SP	**authenticationInd**	Handles an AuthenticationCall by checking the authenticationKey and calling an authenticationResp with the authorization bits if the auhenticationKey is ok. Otherwise a VIException shall be thrown.	
Resp	**authenticationResp**		
Parameter	**Name**	**Type**	**Description**
	`call`	`AuthenticationCall`	A java representation of the ASN.1 AuthenticationCall object.

API	`UGPPackage authenticationInd(int callSequenceNumber, AuthenticationCall call)`
	`  throws VIException;`
	
	`public class AuthenticationCall {`
	`  String authenticationKey;`
	`}`		

Impl	`AuthorizationBits authorization = check(call.getAuthenticationKey());` `return authenticationResp(callSequenceNumber, authorization);`		

8.1.3 getSupportedDataInd

Table 29 defines the getSupportedData indication service.

Table 29 — Definition of the getSupportedData indication service

SP	**getSupportedDataInd**	Handles a GetSupportedDataCall by interpreting the call parameters and calling a getSupportedDataResp with the corresponding data parameters. Otherwise a VIException shall be thrown.	
Resp	**getSupportedDataResp**		
Parameter	**Name**	**Type**	**Description**
	call	GetSupportedDataCall	A java representation of the ASN.1 GetSupportedDataCall object.

API	`UGPPackage getSupportedDataInd(int callSequenceNumber, GetSupportedDataCall call)`
	`    throws VIException;`
	
	`public class GetSupportedDataCall {`
	`   SupportedDataFilter supportedDataFilter;`
	`   Vector<Integer>     ecuList;`
	`   AccessType          accessType;`
	`   Integer             dataParamProperty;`
	`}`		

Impl ex-am-ple	`if (call.getSupportedDataFilter() =  = without_ecu_data) {`
	`    Vector<Integer> dataParams = new Vector<Integer>();`
	`    for (DataParam dp: getAllDataParams()) {`
	`        if (matches(dp, call)) {`
	`            dataParams.add(dp.getRvId());`
	`        }`
	`   }`
	`    return getSupportedDataResp(callSequenceNumber, dataParams);`
	`} else {`
	`    ...`
	`}`		

8.1.4 getValueInd

Table 30 defines the getValue indication service.

Table 30 — Definition of the getValue indication service

SP	getValueInd	Handles a GetValueCall by interpreting the call parameters and calling a getValueResp with the corresponding data parameter values. Otherwise a VIException shall be thrown.	
Resp	getValueResp		
Parameter	Name	Type	Description
	call	GeValueCall	A java representation of the ASN.1 GetValueCall object.

API	`UGPPackage getValueInd(int callSequenceNumber, GetValueCall call)`
	`  throws VIException;`
	
	`public class GetValueCall {`
	`  Integer  testInterval;`
	`  Vector<Integer>  dataParamList;`
	`  Vector<DataParamMapping>  dataParamMapping;`
	`  ComplexCondition  condition;`
	`}`		

Impl ex-ample	`Vector<DataParamValueTS> valuesTSs = new Vector<DataParamValueTS>();`
	`//Retrieve all data parameter values with time stamp of the`
	`//requested data parameters`
	`...`
	`if (call.getTestInterval() > 0) {`
	`  //Start scheduled asynchronous job to get the requested data parameter values`
	`}`
	`return getValueResp(callSequenceNumber, valueTSs);`		

8.1.5 setValueInd

Table 31 defines the setValue indication service.

Table 31 — Definition of the setValue indication service

SP	setValueInd	Handles a SetValueCall by interpreting the call parameters and calling a positiveRes. Otherwise a VIException shall be thrown.	
Resp	positiveResp		
Parameter	Name	Type	Description
	call	SetValueCall	A java representation of the ASN.1 SetValueCall object.

API	`UGPPackage setValueInd(int callSequenceNumber, SetValueCall call)`
	`  throws VIException;`
	
	`public class SetValueCall {`
	`  Vector<DataParamValueMapping> valueMapping;`
	`}`		

Impl ex.	`//Execute a job to set the values of the requested data parameters`
	`...`
	`return positiveResp(callSequenceNumber);`		

8.1.6 controlValueInd

Table 32 defines the controlValue indication service.

Table 32 — Definition of the controlValue indication service

SP	controlValueInd	Handles a GetValueCall by interpreting the call parameters and calling a getValueResp with the corresponding data parameter values. Otherwise a VIException shall be thrown.	
Resp	controlValueResp		
Parameter	**Name**	**Type**	**Description**
	call	GetValueCall	A java representation of the ASN.1 GetValueCall object.

API	`UGPPackage getValueInd(int callSequenceNumber, GetValueCall call)`
	`  throws VIException;`
	
	`public class GetValueCall {`
	`  Integer  testInterval;`
	`  Vector<Integer>  dataParamList;`
	`  Vector<DataParamMapping>  dataParamMapping;`
	`  ComplexCondition  condition;`
	`}`		

Impl example	`Vector<DataParamValueTS> valuesTSs = new Vector<DataParamValueTS>();`
	`//Retrieve all data parameter values with time stamp of the`
	`//requested data parameters`
	`...`
	`if (call.getTestInterval() > 0) {`
	`  //Start scheduled asynchronous job to get the requested data parameter values`
	`}`
	`return getValueResp(callSequenceNumber, valueTSs);`		

8.1.7 getDtcInfoInd

Table 33 defines the getDtcInfo indication service.

Table 33 — Definition of the getDtcInfo indication service

SP	**getDtcInfoInd**	Handles a GetDtcInfoCall by interpreting the call parameters and calling a getDtcInfoResp with the corresponding DTC information. Otherwise a VIException shall be thrown.	
Resp	**getDtcInfoResp**		
Parameter	**Name**	**Type**	**Description**
	call	GetDtcInfoCall	A java representation of the ASN.1 GetDtcInfoCall object.

API	`UGPPackage getDtcInfoInd(int callSequenceNumber, GetDtcInfoCall call)`
	`  throws VIException;`
	
	`public class GetDtcInfoCall {`
	`  Integer  testInterval;`
	`  Integer  rdtcBaseId;`
	`  Integer  rdtcSymptomId;`
	`  Vector<Integer>  ecuList;`
	`  Boolean  withEnvData;`
	`  ComplexCondition  condition;`
	`}`		

Impl example	`Vector<DtcInfo> dtcInfos = new Vector<DtcInfo>();`
	`//Retrieve all requested DTC infos`
	`...`
	`if (call.getTestInterval() > 0) {`
	`  //Start scheduled asynchronous job to get the requested DTC infos`
	`}`
	`return getDtcInfoResp(callSequenceNumber, dtcInfos);`		

8.1.8 clearDtcInfoInd

Table 34 defines the clearDtcInfo indication service.

Table 34 — Definition of the clearDtcInfo indication service

SP	**clearDtcInfoInd**	Handles a ClearDtcInfoCall by interpreting the call parameters and calling a positiveResp. Otherwise a VIException shall be thrown.	
Resp	**positiveResp**		
Parameter	**Name**	**Type**	**Description**
	call	ClearDtcInfoCall	A java representation of the ASN.1 ClearDtcInfoCall object.

API	`UGPPackage clearDtcInfoInd(int callSequenceNumber, ClearDtcInfoCall call)`
	`  throws VIException;`
	
	`public class ClearDtcInfoCall {`
	`  Vector<Integer> ecuList;`
	`}`		

Impl ex.	`//Execute a job to clear the requested DTCs`
	`...`
	`return positiveResp(callSequenceNumber);`		

8.1.9 enablePassThruInd

Table 35 defines the enablePassThru indication service.

Table 35 — Definition of the enablePassThru indication service

SP	enablePassThruInd	Handles an EnablePassThruCall by interpreting the call parameters and calling a positiveResp. Otherwise a VIException shall be thrown.	
Resp	positiveResp		
Parameter	Name	Type	Description
	call	EnablePassThruCall	A java representation of the ASN.1 EnablePassThruCall object.

API	UGPPackage enablePassThroughInd(int callSequenceNumber, EnablePassThruCall call)
	throws VIException;
	
	public class EnablePassThruCall {
	String label;
	String key;
	}		

Implex.	if (call.getKey ! = null) { //enable
	//Create a VPN to allow pass thru
	...
	} else { //disable
	//Delete VPN for pass thru
	...
	}
	return positiveResp(callSequenceNumber);		

8.1.10 listFileInd

Table 36 defines the listFile indication service.