

---

---

**Road vehicles — Modular vehicle  
communication interface (MVCI) —**

**Part 2:  
Diagnostic protocol data unit (D-PDU  
API)**

*Véhicules routiers — Interface de communication modulaire du  
véhicule (MVCI) —*

*Partie 2: Interface de programmation d'application d'unité de  
données du protocole de diagnostic (D-PDU API)*

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017



STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017



**COPYRIGHT PROTECTED DOCUMENT**

© ISO 2017, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

[STANDARDSISO.COM](https://standardsiso.com) : Click to view the full PDF of ISO 22900-2:2017

## Contents

1	Scope.....	1
2	Normative references.....	1
3	Terms, definitions and abbreviated terms.....	1
3.1	Terms and definitions.....	1
3.2	Abbreviated terms.....	2
4	Specification release version information .....	4
4.1	Specification release version location .....	4
4.2	Specification release version .....	5
5	Modular VCI use cases.....	5
5.1	OEM merger.....	5
5.2	OEM cross vehicle platform ECU(s).....	5
5.3	Central source diagnostic data and exchange during ECU development.....	5
5.4	OEM franchised dealer and aftermarket service outlet diagnostic tool support.....	6
6	Modular VCI software architecture.....	6
6.1	Overview.....	6
6.2	Modular VCI D-Server software .....	7
6.3	Runtime format based on ODX.....	7
6.4	MVCI protocol module software .....	8
6.5	MVCI protocol module configurations.....	8
7	D-PDU API use cases .....	9
7.1	Overview.....	9
7.2	Use case 1: Single MVCI protocol module.....	9
7.3	Use case 2: Multiple MVCI protocol modules supported by same D-PDU API implementation .....	10
7.4	Use case 3: Multiple MVCI protocol modules supported by different D-PDU API implementations.....	11
8	Diagnostic protocol data unit (D-PDU) API.....	12
8.1	Software requirements.....	12
8.1.1	General requirements.....	12
8.1.2	Vehicle protocol requirements .....	13
8.1.3	Timing requirements for protocol handler messages.....	13
8.1.4	Serialization requirements for protocol handler messages.....	14
8.1.5	Compatibility requirements.....	16
8.1.6	Timestamp requirements .....	16
8.2	API function overview and communication principles .....	16
8.2.1	Terms used within the D-PDU API.....	16
8.2.2	Function overview.....	17
8.2.3	General usage.....	18
8.2.4	Asynchronous and synchronous communication .....	21
8.2.5	Usage of resource locking and resource unlocking .....	21
8.2.6	Usage of ComPrimitives.....	21
8.3	Tool integration .....	36
8.3.1	Requirement for generic configuration .....	36
8.3.2	Tool integrator — Use case.....	36
8.4	API functions — Interface description.....	38
8.4.1	Overview.....	38
8.4.2	PDUConstruct.....	38
8.4.3	PDUDestruct.....	39
8.4.4	PDUIoCtl.....	40

8.4.5	PDUGetVersion .....	42
8.4.6	PDUGetStatus.....	42
8.4.7	PDUGetLastError.....	43
8.4.8	PDUGetResourceStatus .....	44
8.4.9	PDUCreateComLogicalLink.....	45
8.4.10	PDUDestroyComLogicalLink .....	48
8.4.11	PDUConnect .....	49
8.4.12	PDUDisconnect.....	51
8.4.13	PDUlockResource.....	52
8.4.14	PDUUnlockResource .....	53
8.4.15	PDUGetComParam .....	54
8.4.16	PDUSetComParam.....	61
8.4.17	PDUStartComPrimitive.....	63
8.4.18	PDUCancelComPrimitive .....	67
8.4.19	PDUGetEventItem .....	68
8.4.20	PDUDestroyItem.....	69
8.4.21	PDURegisterEventCallback.....	70
8.4.22	EventCallback prototype .....	72
8.4.23	PDUGetObjectid .....	73
8.4.24	PDUGetModuleIds .....	74
8.4.25	PDUGetResourceIds .....	76
8.4.26	PDUGetConflictingResources .....	77
8.4.27	PDUGetUniqueRespIdTable.....	78
8.4.28	PDUSetUniqueRespIdTable .....	79
8.4.29	PDUModuleConnect.....	84
8.4.30	PDUModuleDisconnect.....	86
8.4.31	PDUGetTimestamp .....	87
8.5	I/O control section.....	88
8.5.1	IOCTL API command overview .....	88
8.5.2	PDU_IOCTL_RESET .....	90
8.5.3	PDU_IOCTL_CLEAR_TX_QUEUE .....	91
8.5.4	PDU_IOCTL_SUSPEND_TX_QUEUE.....	91
8.5.5	PDU_IOCTL_RESUME_TX_QUEUE .....	92
8.5.6	PDU_IOCTL_CLEAR_RX_QUEUE .....	92
8.5.7	PDU_IOCTL_CLEAR_TX_QUEUE_PENDING .....	93
8.5.8	PDU_IOCTL_READ_VBATT.....	93
8.5.9	PDU_IOCTL_SET_PROG_VOLTAGE.....	94
8.5.10	PDU_IOCTL_READ_PROG_VOLTAGE.....	95
8.5.11	PDU_IOCTL_GENERIC.....	95
8.5.12	PDU_IOCTL_SET_BUFFER_SIZE .....	96
8.5.13	PDU_IOCTL_GET_CABLE_ID.....	96
8.5.14	PDU_IOCTL_START_MSG_FILTER.....	97
8.5.15	PDU_IOCTL_STOP_MSG_FILTER.....	98
8.5.16	PDU_IOCTL_CLEAR_MSG_FILTER .....	99
8.5.17	PDU_IOCTL_SET_EVENT_QUEUE_PROPERTIES.....	99
8.5.18	PDU_IOCTL_SEND_BREAK .....	100
8.5.19	PDU_IOCTL_READ_IGNITION_SENSE_STATE .....	101
8.5.20	PDU_IOCTL_VEHICLE_ID_REQUEST .....	102
8.5.21	PDU_IOCTL_SET_ETH_SWITCH_STATE.....	103
8.5.22	PDU_IOCTL_GET_ENTITY_STATUS.....	104
8.5.23	PDU_IOCTL_GET_DIAGNOSTIC_POWER_MODE.....	105
8.5.24	PDU_IOCTL_GET_ETH_PIN_OPTION .....	105
8.6	API functions — Error handling .....	106
8.6.1	Synchronous error handling .....	106
8.6.2	Asynchronous error handling.....	106
8.7	Installation .....	106
8.7.1	Generic description .....	106
8.7.2	Windows installation process.....	107

8.7.3	Linux installation process.....	108
8.7.4	Selecting MVCI protocol modules.....	108
8.8	Application notes.....	108
8.8.1	Interaction with the MDF .....	108
8.8.2	Accessing additional hardware features for MVCI protocol modules .....	108
8.8.3	Documentation and information provided by MVCI protocol module vendors.....	109
9	Using the D-PDU API with existing applications.....	109
9.1	SAE J2534-1 and RP1210a existing standards .....	109
10	Data structures.....	110
10.1	API functions — Data structure definitions .....	110
10.1.1	Abstract basic data types .....	110
10.1.2	Definitions.....	111
10.1.3	Bit encoding for UNUM32.....	111
10.1.4	API data structures.....	111
Annex A	(normative) D-PDU API compatibility mappings .....	125
A.1	Mapping of D-PDU API and D-Server API .....	125
A.2	Mapping of D-PDU API and ODX.....	125
Annex B	(normative) D-PDU API standard ComParams and protocols.....	126
B.1	Standardized protocols — Support and naming conventions.....	126
B.1.1	General.....	126
B.1.2	SAE J2534 and RP1210a standard protocol names.....	126
B.1.3	Protocol names — Combination list.....	126
B.1.4	Standard protocol naming guidelines.....	128
B.1.5	Standard protocol short names.....	128
B.1.6	D-PDU API optional OBD protocol short names.....	129
B.2	Standard protocol pin types and short names .....	129
B.3	Standard protocol communication parameters (ComParams) .....	130
B.3.1	Protocol ComParam description method .....	130
B.3.2	ComParam class .....	131
B.3.3	ComParam data type.....	132
B.3.4	ComParam support .....	135
B.3.5	ComParam usage .....	135
B.3.6	ComParam OSI layer reference .....	136
B.4	ComParam summary tables .....	136
B.4.1	Application layer.....	136
B.4.2	Transport layer .....	140
B.4.3	Physical layer .....	148
B.4.4	CAN identifier format for ISO 15765 and ISO 11898 protocols .....	150
B.4.5	Non-CAN ComParam examples.....	159
B.4.6	29-bit CAN identifier data page bits.....	160
B.5	ComParam detailed descriptions .....	160
B.5.1	ComParam definitions for application layer .....	160
B.5.2	ComParam definitions for transport layer.....	176
B.5.3	ComParam definitions for physical layer .....	199
Annex C	(informative) D-PDU API manufacturer-specific ComParams and protocols.....	203
C.1	Manufacturer-specific protocols — Support and naming conventions .....	203
C.1.1	General.....	203
C.1.2	Manufacturer protocol naming guidelines.....	203
C.1.3	Manufacturer protocol communication parameters (ComParams) .....	203
Annex D	(normative) D-PDU API constants.....	205
D.1	Constants.....	205
D.1.1	D-PDU API item type values .....	205
D.1.2	ComPrimitive type values .....	205
D.1.3	Object type values.....	206
D.1.4	Status code values .....	206

D.1.5	Information event values .....	207
D.1.6	Resource status values .....	208
D.1.7	Resource lock values.....	208
D.1.8	Event callback data values .....	208
D.1.9	Reserved ID and handle values.....	209
D.1.10	IOCTL filter types values.....	209
D.1.11	IOCTL event queue mode type values.....	209
D.2	Flag definitions .....	210
D.2.1	TxFlag definition .....	210
D.2.2	RxFlag definition .....	211
D.2.3	ClICreateFlag definition .....	212
D.2.4	TimestampFlag definition .....	213
D.3	Function return values.....	214
D.4	Event error codes.....	216
D.4.1	Error event code returned in PDU_IT_ERROR.....	216
D.4.2	Additional error code returned in PDU_IT_ERROR.....	216
Annex E (normative)	Application defined tags.....	219
Annex F (normative)	RDF and MDF description files .....	220
F.1	D-PDU API root description file (RDF) .....	220
F.1.1	General .....	220
F.1.2	UML diagram of RDF.....	220
F.2	MVCI module description file (MDF).....	221
F.2.1	General .....	221
F.2.2	ComParam string format .....	221
F.2.3	ComParam resolution tag.....	224
F.2.4	UML diagram of MDF.....	224
F.2.5	UML diagram of MDF elements COMPARAM .....	224
F.2.6	UML diagram of MDF element RESOURCE.....	225
F.2.7	UML diagram of MDF element MODULETYPE .....	226
F.3	Cable description file (CDF) .....	226
F.3.1	General .....	226
F.3.2	UML diagram of CDF .....	226
F.4	XML schema .....	227
F.5	Description file examples .....	232
F.5.1	Example MVCI protocol module .....	232
F.5.2	Example root description file.....	234
F.5.3	Example module description file .....	235
F.5.4	Example cable description file.....	276
Annex G (informative)	Resource handling scenarios .....	278
G.1	Resource handling at the API level.....	278
G.1.1	Obtaining resource and object ids.....	278
G.1.2	Example MVCI protocol module resource selection .....	281
Annex H (informative)	D-PDU API partitioning.....	283
H.1	Functional partitioning of a D-PDU API .....	283
H.1.1	ODX data base.....	283
H.1.2	MVCI D-Server .....	283
H.1.3	VCI protocol module.....	283
H.1.4	Vehicle bus network.....	285
Annex I (informative)	Use case scenarios .....	286
I.1	Negative response handling scenarios.....	286
I.1.1	General .....	286
I.1.2	Physical addressing.....	286
I.1.3	Functional addressing .....	291
I.2	ISO 14229-1 UDS.....	298
I.2.1	Suppress positive response scenarios .....	298

I.2.2	Service 0x2A use case scenario .....	303
I.3	Service shop use case scenario.....	306
I.4	Vehicle bus baud rate changing scenario.....	307
I.4.1	General .....	307
I.4.2	Device use .....	307
I.4.3	Example scenarios.....	308
I.5	SAE J1939 use cases .....	310
I.5.1	SAE J1939 CAN ID Formation .....	310
I.5.2	Setting up ComParams for a SAE J1939 ComLogicalLink.....	311
I.5.3	Case 1: Receiving active DTC from DM1 PGN 65226 (0xFECA) .....	312
I.5.4	Case 2: Receive PGN 65264 (0xfef0) — ECU data .....	312
I.5.5	Case 3: Request previously active DTC PGN 65227 (0xFECB) .....	313
I.5.6	Case 4: Read VIN PGN 65260 (0xFEEC).....	314
I.5.7	Case 5: Clear specific DTC (DM22) PGN 49920 (0xC300) .....	314
I.5.8	Case 6: Read VIN in raw mode PGN 65260 (0xFEEC).....	315
I.5.9	Case 7: Data security in raw mode (DM18) PGN 54272 (0xD400).....	316
I.6	Multiple clients use cases.....	317
I.6.1	Definition.....	317
I.6.2	Multiple clients configurations .....	317
I.6.3	Example scenarios.....	321
I.7	P3 sequencing.....	322
Annex J (normative)	OBD protocol initialization .....	324
J.1	OBD application .....	324
J.1.1	OBD concept .....	324
J.1.2	Automatic OBD protocol determination .....	324
J.1.3	Simultaneous protocol scan sequence using the D-PDU API.....	325
Annex K (normative)	DoIP implementation .....	339
K.1	DoIP vehicle architecture.....	339
K.2	DoIP use case definition .....	340
K.2.1	Diagram with use cases and actors .....	340
K.2.2	Use cases.....	340
K.2.3	D-PDU API and D-Server .....	342
K.2.4	MVCI and D-PDU API.....	343
K.2.5	ISO 13400-3 Ethernet pin option determination and activation .....	351
K.3	Handling of the DoIP protocol in D-PDU API.....	352
K.3.1	General .....	352
K.3.2	Overview: Usage of D-PDU API for DoIP protocol .....	352
K.3.3	Socket handling.....	357
K.3.4	D-PDU API DoIP NACK behaviour.....	358

## 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](http://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](http://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](http://www.iso.org/iso/foreword.html).

This document was prepared by ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*.

This second edition cancels and replaces the first edition (ISO 22900-2:2009), which has been technically revised and includes the following changes:

- former corrigendum concerning Kline handling;
- former DoIP amendment;
- how to detect the 2 possible DoIP pin assignment; and
- introduction of CAN-FD.

A list of all parts in the ISO 22900 series can be found on the ISO website.

## Introduction

The ISO 22900 series is applicable to vehicle electronic control module diagnostics and programming.

This document was established in order to more easily exchange software and hardware of vehicle communication interfaces (VCIs) among diagnostic applications. It defines a generic and protocol independent software interface towards the modular vehicle communication interface (MVCI) protocol module, such that a diagnostic application based on this software interface can exchange the MVCI protocol module or add a new MVCI protocol module with minimal effort. Today, the automotive aftermarket requires flexible usage of different protocol modules for vehicles of different brands. Many of today's protocol modules are incompatible with regard to their hardware and software interface, such that, depending on the brand, a different protocol module is required.

The objective of this document is to specify the diagnostic protocol data unit application programming interface (D-PDU API) as a generic software interface and to provide a "plug and play" concept for access onto different MVCI protocol modules from different tool manufacturers. The D-PDU API will address the generic software interface, the protocol abstraction, its exchangeability, as well as the compatibility towards existing standards such as SAE J2534-1 and RP1210a.

The implementation of the modular VCI concept facilitates co-existence and re-use of MVCI protocol modules, especially in the aftermarket. As a result, diagnostic or programming applications can be adapted for different vehicle communication interfaces and different vehicles with minimal effort, thus helping to reduce overall costs for the tool manufacturer and end user.

Vehicle communication interfaces compliant with ISO 22900 series support a protocol-independent D-PDU API as specified in this document.

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

# Road vehicles — Modular vehicle communication interface (MVCI) — Part 2: Diagnostic protocol data unit (D-PDU API)

## 1 Scope

This document specifies the diagnostic protocol data unit application programming interface (D-PDU API) as a modular vehicle communication interface (MVCI) protocol module software interface and common basis for diagnostic and reprogramming software applications.

This document covers the descriptions of the application programming interface (API) functions and the abstraction of diagnostic protocols, as well as the handling and description of MVCI protocol module features. Sample MVCI module description files accompany this document.

The purpose of this document is to ensure that diagnostic and reprogramming applications from any vehicle or tool manufacturer can operate on a common software interface and can easily exchange MVCI protocol module implementations.

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 15765-2, *Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services*

ISO 22900-1, *Road vehicles — Modular vehicle communication interface (MVCI) — Part 1: Hardware design requirements*

SAE J2411, *Single wire CAN network for vehicle applications*

## 3 Terms, definitions and abbreviated terms

For the purposes of this document, the following terms and definitions 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 Terms and definitions

#### 3.1.1

##### **application**

way of accessing the diagnostic protocol data unit application programming interface (D-PDU API)

Note 1 to entry: From the perspective of the D-PDU API, it does not make any difference whether an application accesses the software interface directly or through an MVCI D-Server. Consequently, in this document, the term “application” represents both ways of accessing the D-PDU API.

#### 3.1.2

##### **ComLogicalLink**

logical communication channel towards a single electronic control unit (ECU) or towards multiple electronic control units

#### 3.1.3

##### **COMPARAM-SPEC**

protocol-specific set of predefined communication parameters (ComParams), the value of which can be changed in the context of a layer or specific diagnostic service

Note 1 to entry: This part of the model can also contain OEM-specific ComParams.

### 3.1.4

#### **ComPrimitive**

smallest aggregation of a communication service or function

EXAMPLE A request message to be sent to an ECU.

### 3.1.5

#### **Ethernet**

physical network media type

### 3.1.6

#### **local network**

part of the network connected directly to the tester, also called "primary network" in some subclauses

### 3.1.7

#### **remote network**

part of the network located behind a gateway connected to the tester

Note 1 to entry: Also called "secondary network" in some subclauses.

### 3.1.8

#### **diagnostic communication over Internet Protocol modular vehicle communication interface module**

##### **DoIP MSCI module**

MSCI module able to handle connections to one or multiple DoIP entities

### 3.1.9

#### **diagnostic communication over Internet Protocol gateway**

##### **DoIP gateway**

gateway connected to the tester via DoIP protocol

### 3.1.10

#### **diagnostic communication over Internet Protocol node**

##### **DoIP node**

ECU connected to the tester directly via DoIP protocol

Note 1 to entry: ECU has no gateway capabilities.

### 3.1.11

#### **diagnostic communication over Internet Protocol entity**

##### **DoIP entity**

general term for either a DoIP gateway or a DoIP node

## 3.2 Abbreviated terms

API	application programming interface
ASCII	American Standard for Character Information Interchange
CAN	controller area network
CAN IDs	controller area network identifiers
CDF	cable description file
CLL	ComLogicalLink

ComLogicalLinks	communication logical links
ComParam	communication parameter
ComParamSpec	communication parameter specification
ComPrimitive	communication primitive
CRC	cyclic redundancy check
DLC	data link connector
DLL	dynamic link library
DoIP	diagnostic communication over Internet Protocol
D-PDU	diagnostic protocol data unit
D-Server	diagnostic server
ECU	electronic control unit
HDD	hard disk drive
HI	differential line — high
HW	Hardware
IEEE 1394	firewire serial bus
IFR	in-frame response
IGN	Ignition
IOCTL	input/output control
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
K	UART K-Line
KWP	keyword protocol
L	UART L-Line
LL	Logical Link
LOW	differential line — low
MDF	module description file
MOST	media oriented systems transport
MVCI	modular vehicle communication interface
ODX	open diagnostic data exchange

OEM	original equipment manufacturer
OSI	open systems interconnection
PC	personal computer
PCI	protocol control information
PGN	parameter group number
PROGV	programmable voltage
PWM	pulse width modulation
RDF	root description file
RX	UART uni-directional receive
SCI	serial communications interface
SCP	standard corporate protocol
TCP	transmission control protocol
TX	UART uni-directional transmit
UDS	unified diagnostic services
UDSonCAN	unified diagnostic services on CAN
UDSonIP	unified diagnostic services on IP
USB	universal serial bus
USDT	unacknowledged segmented data transfer
UUDT	unacknowledged un-segmented data transfer
VCI	vehicle communication interface
VPW	variable pulse width
XML	extensible markup language

## 4 Specification release version information

### 4.1 Specification release version location

Specification release version information is contained in each modular VCI release document specification under the same clause title “Specification release version information”. It is important to check for feature support between modular VCI release specifications if the most recent API features shall be implemented. The D-PDU API supports the reading of version information by the API function call PDUGetVersion.

Release version information is also contained in the following files:

- root description file (RDF);
- module description file (MDF);

- cable description file (CDF);
- D-PDU API library file.

## 4.2 Specification release version

The specification release version of this document is 2.2.1.

## 5 Modular VCI use cases

### 5.1 OEM merger

In the past, several OEMs in the automotive industry have merged into one company.

All companies try to leverage existing (legacy) components and jointly develop new products, which are common across different vehicle types and badges.

If OEMs already had modular VCI compliant test equipment, it would be simple to connect MVCI protocol modules from merged OEMs into one chassis or device. All protocols would be supported by a single MVCI protocol module configuration without any replacement of MVCI protocol module hardware at the dealer site. The same applies for engineering and some of this concept might also work for production plants (end of line).

### 5.2 OEM cross vehicle platform ECU(s)

OEMs specify requirements and design electronic systems to be implemented in multiple vehicle platforms in order to avoid re-inventing a system for different vehicles. The majority of design, normal operation and diagnostic data of an electronic system are re-used if installed in various vehicles. The engineering development centres are located worldwide. A great amount of re-authoring of diagnostic data is performed to support different engineering test tools.

Providing diagnostic data in an industry standard format like ODX and XML will avoid re-authoring into various test tool specific formats at different system engineering locations. The D-PDU API supports this re-use concept by fully abstracting vehicle protocols into the industry supported ComParam descriptions.

### 5.3 Central source diagnostic data and exchange during ECU development

Single source origin of diagnostic data (as depicted in Figure 1), combined with a verification and feedback mechanism and distribution to the end users, is highly desirable in order to lower engineering expenses. Engineering, manufacturing and service specify which protocol and data shall be implemented in the ECU. This information will be documented in a structured format like XML. Furthermore, the same structured data files can be used to setup the diagnostic engineering tools to verify proper communication with the ECU and to perform functional verification and compliance testing of the ECU. Once all quality goals are met, these structured data files shall be released to the OEM database. An Open Diagnostic data eXchange (ODX) schema has been developed for the purpose of supporting these structured formatted files used for ECU diagnosis and validation.

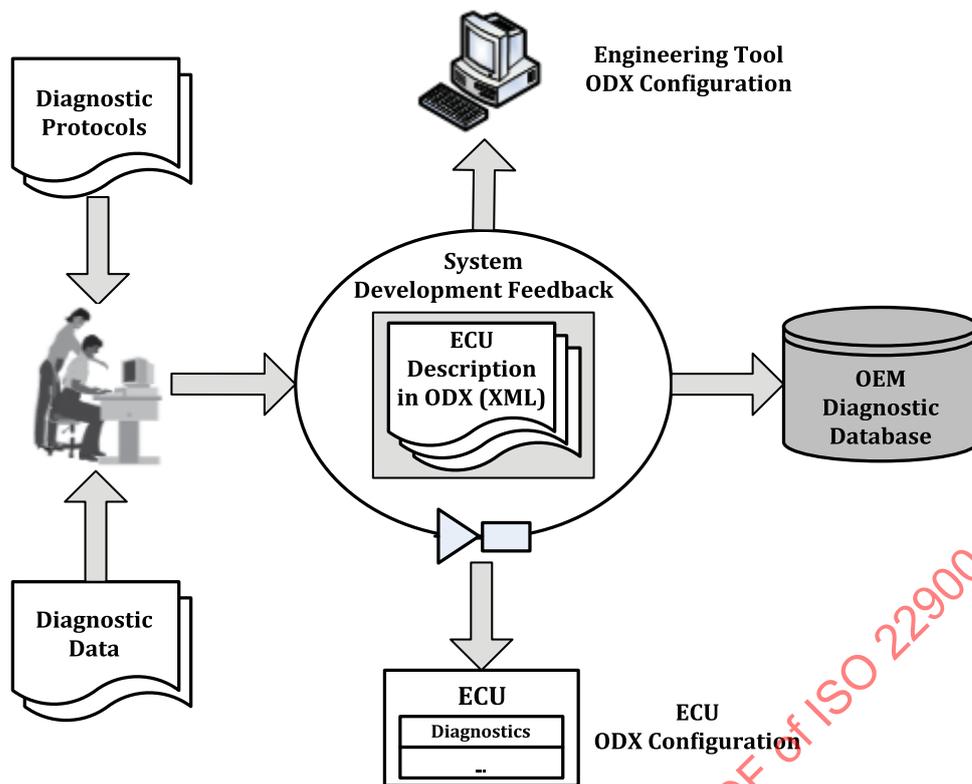


Figure 1 — Example of central source engineering diagnostic data process

#### 5.4 OEM franchised dealer and aftermarket service outlet diagnostic tool support

The service shop uses the modular VCI hardware and software for vehicle diagnosis and enhanced procedural testing. By using the same engineering, manufacturing and service functions as those used for individual ECU testing, the reliability of the data is maintained. A modular VCI protocol module can be used with any PC (handheld or stationary) and can be utilized as an embedded device.

### 6 Modular VCI software architecture

#### 6.1 Overview

The modular VCI concept is mainly based on three software components (see Figure 2):

- MSCI D-Server software;
- runtime data based on ODX;
- MSCI protocol module software.

The application accesses the MSCI D-Server through the MSCI D-Server API. The D-Server obtains all required information about the ECU(s) out of the ODX runtime data. Using the ODX runtime data information, the D-Server converts the application's request into a byte stream, which is called a diagnostic protocol data unit (D-PDU). The D-PDU is handed over to the MSCI protocol module through the D-PDU API. The MSCI protocol module transmits the D-PDU to the vehicle's ECU(s). The other way around, the MSCI protocol module receives the vehicle's response(s) and reports the response data to the D-Server. Again, using the ODX runtime data, the D-Server interprets the D-PDU and provides the interpreted symbolic information to the application.

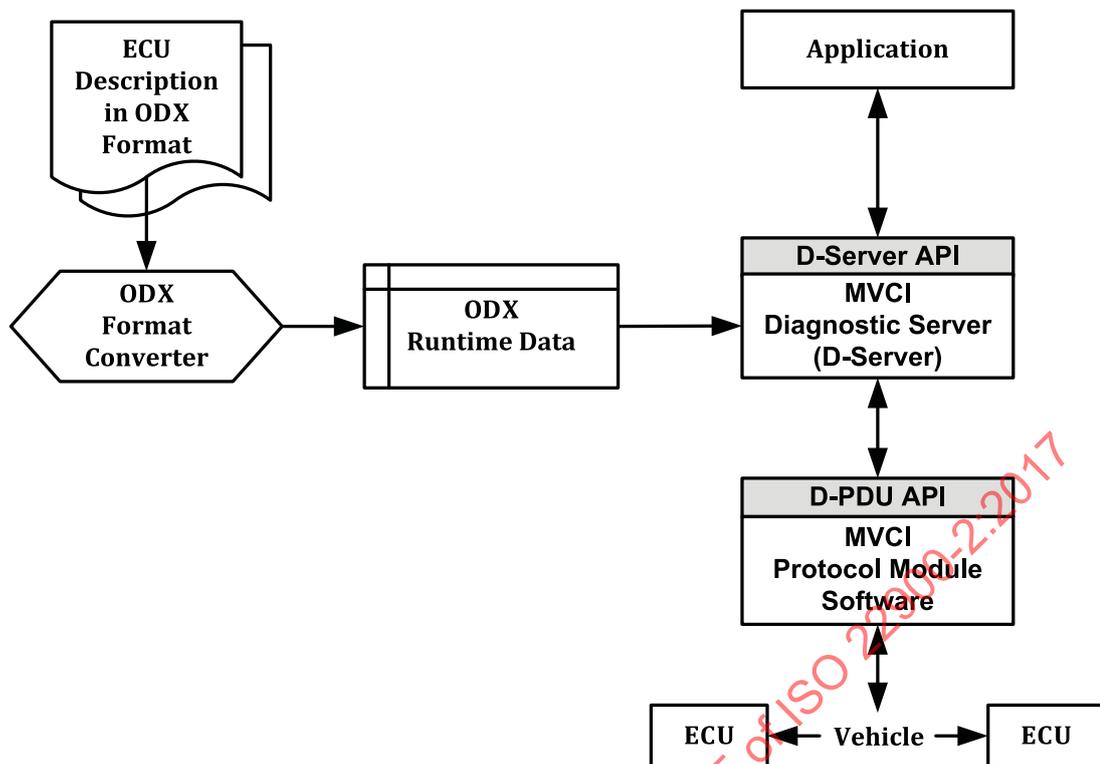


Figure 2 — MVCI software architecture

The grey shading of symbols indicates reference to the following International Standards:

- ODX: ISO 22901-1;
- D-Server API: ISO 22900-3;
- D-PDU API: ISO 22900-2;
- MVCI protocol module: ISO 22900-1.

## 6.2 Modular VCI D-Server software

The MVCI D-Server is accessible through the MVCI D-Server API. By accessing this API, the application may browse the available features for each ECU and initiate a request towards an ECU using simple symbolic expressions. If the request requires input parameters, they can be specified using symbolic expressions as well. The MVCI D-server takes the symbolic request, including input parameters, and converts them into a diagnostic request message as defined at the protocol level. The diagnostic request message represents the diagnostic protocol data unit (D-PDU) as passed to the MVCI protocol module through the D-PDU API. Vice versa, the D-Server converts diagnostic response messages as retrieved from the MVCI protocol module back to symbolic information and provides it to the application.

For a detailed description and the complete MVCI D-Server API definition, see ISO 22900-3.

## 6.3 Runtime format based on ODX

For every conversion from symbolic requests to diagnostic request messages, and vice versa for responses, the MVCI D-Server obtains the required information from the runtime database. The database defines the structure of every diagnostic request and response as supported by an ECU. The database defines byte and bit positions, width and type of every input and output parameter.

Even though the MVCI D-Server obtains its information from a runtime database, the runtime database and format are not specified by the MVCI standard. Instead, the MVCI standard defines an exchange format to import and export the ECU description across OEMs and tool suppliers. The runtime format is left up to the system designers.

The exchange format is called open diagnostic data exchange format (ODX format). For a detailed description, see ISO 22901-1.

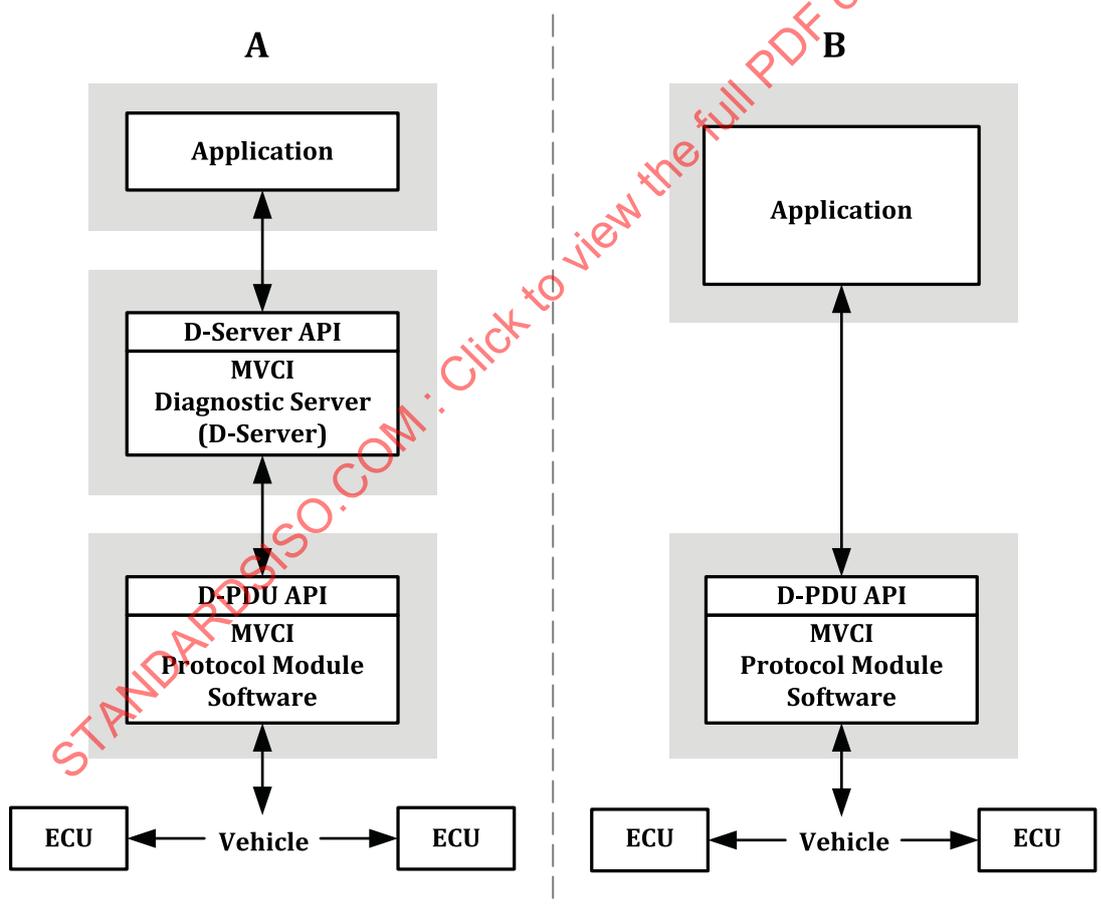
### 6.4 MVCI protocol module software

The MVCI protocol module is accessible through the D-PDU API. The application issues diagnostic requests through the D-PDU API. The MVCI protocol module takes the request D-PDU and transmits it to the vehicle's ECU(s) according to the vehicle communication protocol. Header type, checksum information and D-PDU segmentation depend on the vehicle communication protocol and shall be handled transparently by the MVCI protocol module. Also, the MVCI protocol module observes the timing between message frames and requests and responses on the physical interfaces. After completion, the MVCI protocol module simply has to deliver the response back to the application or report an error condition.

### 6.5 MVCI protocol module configurations

The D-PDU API and MVCI protocol modules work in many configurations. A MVCI D-Server is not required as the application interface to the D-PDU API.

Figure 3 shows two such configurations to point out the differences.



- Key**  
 A application with MVCI D-Server  
 B application without MVCI D-Server

**NOTE** From the perspective of the D-PDU API, it does not make a difference whether an application accesses the software interface directly or through an MVCI D-Server. Consequently, in this document, the term "application" represents both ways of accessing the D-PDU API.

Figure 3 — MVCI configurations

## 7 D-PDU API use cases

### 7.1 Overview

The MVCI protocol module is the key component to exchange implementations of diagnostic protocols among OEMs and tool suppliers without re-engineering already implemented software. By relying on the D-PDU API, the application may easily access other or additional MVCI protocol module implementations. In a similar way to existing standards such as SAE J2534-1 and RP1210a, applications compliant to the MVCI standard are basically independent of the underlying device as long as the required physical interface is supported and no tool supplier specific feature is required.

Even though the D-PDU API extends the capabilities beyond the definitions of SAE J2534-1 and RP1210a, the existing standards and their related devices and applications do not become obsolete by introducing the D-PDU API. Instead, the transition and co-existence of all standards are facilitated to save development and investment costs. The definition of the D-PDU API is closely related to SAE J2534-1 and RP1210a to allow mapping of functionality in both directions. However, it extends their definitions to cover the full width of enhanced diagnostics.

The fulfilment of the following use cases is crucial for the inter-exchange of protocol module implementations according to MVCI, SAE J2534-1 and RP1210a.

NOTE In the use case figures (see Figures 4 to 6), the grey boxes suggest a specific software component architecture. This representation is not intended to be construed as the only possible architectural solution. Depending on the situation, there can be more software components or fewer software components.

### 7.2 Use case 1: Single MVCI protocol module

The single MVCI protocol module configuration (see Figure 4) is the simplest configuration where the D-PDU API implementation and the MVCI protocol module hardware are obtained from the same vendor. The application will access the single MVCI protocol module through a single D-PDU API. Parallel access onto multiple D-PDU APIs is not required. Resource handling is completely covered inside the D-PDU API implementation.

This use case applies to basically all stand-alone MVCI protocol module device configurations.

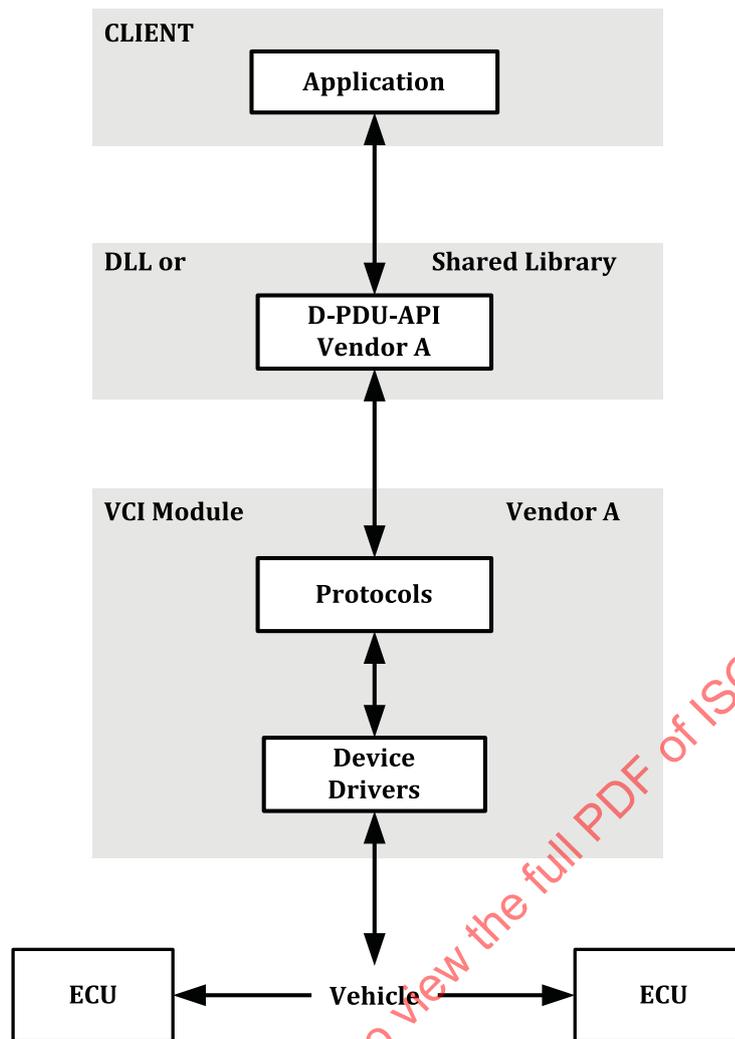


Figure 4 — MVCI configuration with single MVCI protocol module

**7.3 Use case 2: Multiple MVCI protocol modules supported by same D-PDU API implementation**

There are different configurations with multiple MVCI protocol modules. In this use case, a D-PDU API implementation may support more than one MVCI protocol module at a time, where both modules and D-PDU API implementations are from a single vendor (see Figure 5). The application will access the MVCI protocol modules through a single D-PDU API. Parallel access onto multiple D-PDU APIs is not required. However, the application may access and operate the MVCI protocol modules at the same time in parallel if the D-PDU API implementation provides the capabilities. Resource handling is completely covered inside the D-PDU API implementation.

This use case applies to MVCI protocol module device configurations where the vendor integrates support for multiple MVCI protocol modules into one software package.

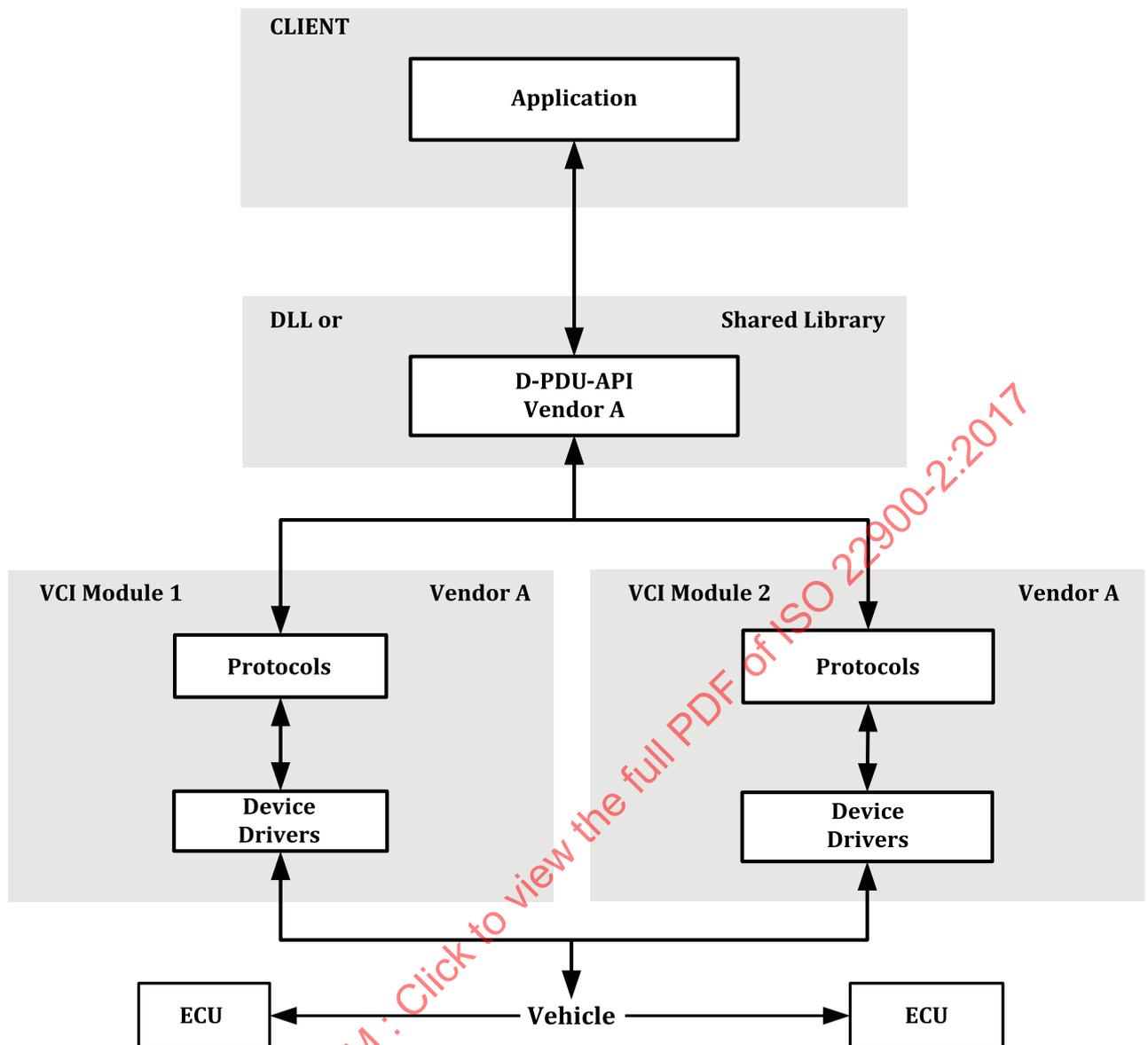


Figure 5 — Multiple MVCI protocol modules supported by same D-PDU API implementation

#### 7.4 Use case 3: Multiple MVCI protocol modules supported by different D-PDU API implementations

In most cases, when combining MVCI protocol modules of different suppliers, the MVCI protocol modules are not accessed through the same D-PDU API implementation (see Figure 6). Neither of the implementations knows about the other suppliers' MVCI protocol modules. It cannot communicate with them, since the D-PDU API does not define an explicit interface hardware type, nor the communication protocol on the interface. Therefore, MVCI protocol modules of different suppliers will be addressed through separate D-PDU API implementations. Each D-PDU API implementation may support more than one MVCI protocol module at a time and more than one D-PDU API implementation may co-exist on the same runtime environment at the same time.

The application may access multiple D-PDU APIs (and their MVCI protocol modules) in parallel, if it needs to use resources of more than one D-PDU API. As a result, each D-PDU API implementation shall be able to run concurrently with other D-PDU API implementations. As in use case 2, resource handling is completely covered inside the D-PDU API implementation with respect to one implementation. As use case 3 assumes multiple D-PDU API implementations not knowing each other, the application is required to handle the resources across D-PDU API implementations.

This use case applies to MVCI protocol module device configurations where a tool supplier integrates support for multiple MVCI protocol modules of different vendors into one software package.

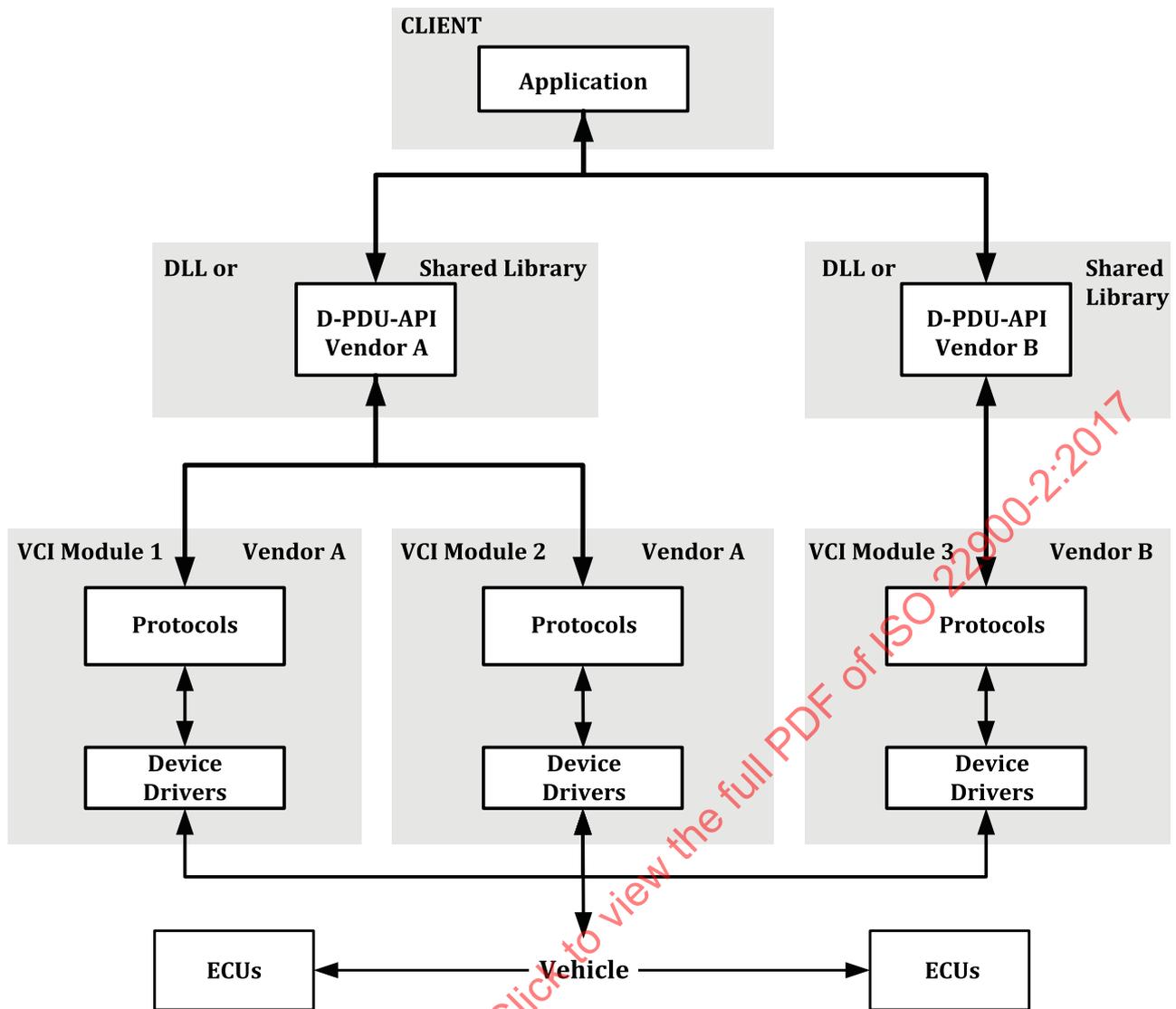


Figure 6 — Multiple MSCI protocol modules supported by different D-PDU API implementations

## 8 Diagnostic protocol data unit (D-PDU) API

### 8.1 Software requirements

#### 8.1.1 General requirements

The MSCI devices shall be accessed through dynamically linkable software modules, i.e. dynamic link libraries for Windows systems (DLLs) and/or library modules for Linux systems. The linkable software module will be referred to as the D-PDU API implementation. It queries the available MSCI protocol modules, takes care of device identification (e.g. reading firmware version, etc.) and low-level communication with all MSCI protocol modules supported by the respective library. However, the device query and identification does not contain any functionality of the system level driver, as it is required for USB, Ethernet and wireless communication. The system low-level driver is responsible for the detection and enumeration of the device interface. The system level driver is considered a part of the delivery from the MSCI protocol module supplier, but its interface is proprietary and not part of this document.

Multiple MSCI protocol modules may be separately accessible by the application using the same D-PDU API implementation. This could typically apply to MSCI protocol module variants from a single supplier. However, MSCI protocol modules may also be accessible through separate D-PDU API implementations. Such cases would typically occur when combining multiple MSCI protocol modules from different suppliers.

In order to declare the capabilities of a D-PDU API implementation, a MVCI protocol module vendor shall provide a module description file (MDF) in XML. The MDF shall contain all information about supported MVCI protocol module types, bus types, protocols and communication links, as well as all related information regarding parameters and I/O controls. The application may parse the file for resources and use them dynamically. It may also make use of static information. In the latter case, the application developer could create a C header file containing and statically matching all necessary resource IDs. For a detailed description and structure of the files, see F.2.

All API functions return a SNUM32 value representing the function result.

The D-PDU API implementation shall not be restricted to a dedicated operating system or programming language and shall be portable. However, for unique and clear definition, C was chosen as the programming language to describe the API.

In general, the D-PDU API implementation shall be made available as a dynamically linkable software module independent of the target operating system. The approach of a separate software module guarantees easy exchange. However, in some cases, it may be useful or more appropriate to link the software module statically. Those cases are considered as proprietary solutions and shall not be the main target of this document.

The D-PDU API implementation shall support, at a minimum, single clients and asynchronous, multi-thread operation. Multi-client support is not a requirement, but may be offered as an additional feature by an MVCI protocol module vendor. A multi-client implementation shall support multiple sessions and links in parallel. For every communication link, the implementation shall take care of queuing communication requests.

The D-PDU API implementation shall cover full duplex and event-driven communication, enabling coverage of advanced vehicle communication principles (e.g. response on event, periodic transmission, etc).

### 8.1.2 Vehicle protocol requirements

The D-PDU API functions shall be protocol independent. Since protocol standards have frequently changed in the past and new protocols will be released in the future, the D-PDU API needs to be flexible and generic enough to easily cover additional protocols not taken into account at time of definition. In order to provide the application the capability to use any protocol and any of its tool supplier specific implementations, all protocol-related ComParams have to be documented in a standardized and generic manner. The documentation is stored in a module description file (MDF) in XML and shall be provided by the MVCI protocol module supplier.

In general, there is no minimum set of protocols with respect to the D-PDU API. However, in order to provide the required SAE J2534-1 and RP1210a compatibility, and to avoid interference of D-PDU API implementations of different suppliers, minimal protocol naming conventions are necessary.

### 8.1.3 Timing requirements for protocol handler messages

There can be unexpected results if two requests are made to the same ECU "simultaneously". Most ECUs will ignore the second request, but some ECUs will ignore both requests. As a result, the protocol handler has to properly serialize requests across multiple CLLs, while still allowing valid parallel communication.

It should also be emphasized that for the cases where CLLs are sharing a physical serial bus, all timing requirements (CP\_P3Min, CP\_P3Phys and CP\_P3Func) shall be satisfied before subsequent transmissions can occur (see I.7).

Message serialization is also required for some complex single CLL scenarios. Consider the case where Tester Present messages, functionally addressed ComPrimitives and physically addressed ComPrimitives are all occurring on one bus. Message serialization may be required to assure that the protocol handler adheres to proper inter-transmit timing and receive timing. All standard protocols have their timing individually defined using ComParams. This allows for minor changes in the timing behaviour of a protocol in order to satisfy the unique attributes of an installed vehicle ECU.

For certain messages, the ability to be able to send the request immediately is desirable. One such message would be a Tester Present where it is undesirable that they are prevented from being sent by a previous request taking too long. This can happen when a request is extended by a 0x78 pending response. It is possible to send these messages so long as sending them does not cause undesirable effects. Table 1 shows the possible scenarios where

requests can be sent immediately following a previous request and when the previous request shall be allowed to complete or timing applied.

Timing between message requests on the serial bus is the responsibility of the MSCI Protocol Handlers. The MSCI protocol module shall ensure that the inter-message timing is correctly managed (see Table 1).

**Table 1 — Functional and physical address handling**

Previous request type	Current request type	Handling in the tester
Physical with response	physical with or without response	Wait until the completion of the previous physical request (positive response or negative response other than 0x78) before transmitting a new physical request. There is no timing applied to this rule, the tester can transmit the physically addressed request immediately after the previous physically addressed service has been finished (final response received).
	functional with response	Wait until the completion of the previous physical request (positive response or negative response other than 0x78) before transmitting the functional request that requires a response. There is no timing applied to this rule, the tester can transmit the functionally addressed request immediately after the previous physically addressed service has been finished (final response received).
	functional without response	The tester is allowed to transmit the functional request that does not require a response immediately (e.g. any physical request followed by a functional Tester Present). NOTE For physical cyclic responses, the tester waits for the first response before sending a functional Tester Present).
Physical without response	physical with or without response	Wait CP_P3Phys after the completion of the previous physical request without response before transmitting a new physical request.
	functional with response	Wait CP_P3Phys after the completion of the previous physical request without response before transmitting a new functional request.
	functional without response	The tester is allowed to transmit the functional request that does not require a response immediately (e.g. any physical request followed by a functional Tester Present).
Functional with response	physical with or without response	Wait until the completion of the previous functional request that requires a response (positive response or negative response other than 0x78) before transmitting the physical request. There is no timing applied to this rule, the tester can transmit the physically addressed request immediately after the previous functionally addressed service has been finished (final response received).
	functional with or without response	Wait until the completion of the previous functional request that requires a response (positive response or negative response other than 0x78) before transmitting the next functional request that does not require a response. There is no timing applied to this rule, the tester can transmit the functionally addressed request immediately after the previous functionally addressed service has been finished (final response received).
Functional without response	physical with or without response	The tester is allowed to transmit the physical request immediately (e.g. functional Tester Present followed by any physical request).
	functional with or without response	Wait CP_P3Func after the transmission of the functional request that does not require a response before transmitting the functional request that does or does not require a response.

**8.1.4 Serialization requirements for protocol handler messages**

Many vehicle serial bus protocols require serialization of messages sent on the bus, e.g. sending functional and physical messages on a shared physical serial bus can be accomplished if the physically addressed ECU is not in the functional group (i.e. a group of ECUs which can be addressed with the same functional address). For this case, the CLLs share the ECU being addressed and the messages/frames need to be serialized. See Figure 7.

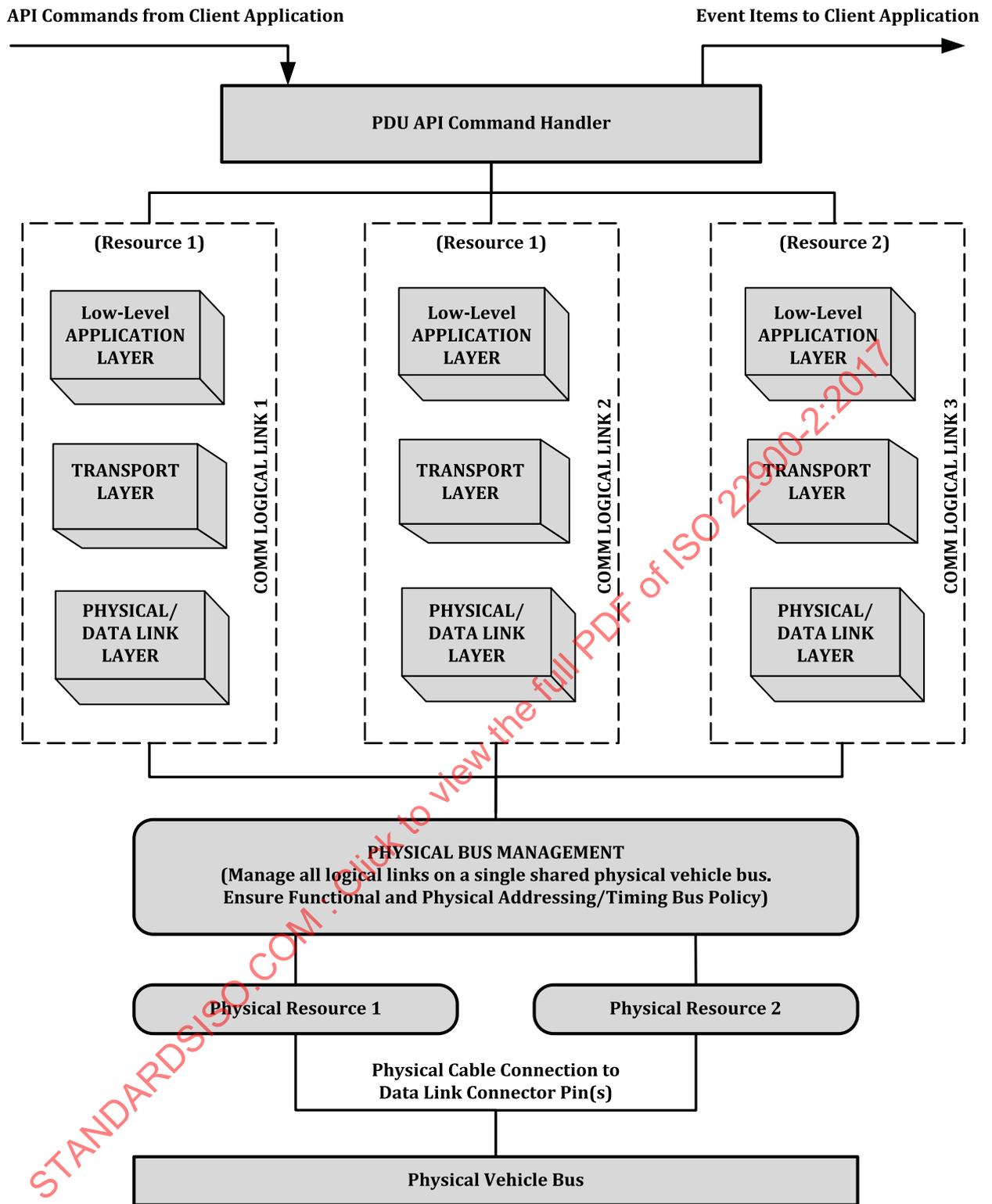


Figure 7 — CLLs sharing physical bus with message serialization example

## 8.1.5 Compatibility requirements

### 8.1.6 Timestamp requirements

#### 8.1.6.1 General

This subclause describes the requirements of the timestamp mechanism that shall be used for the D-PDU API.

The unit of all timestamps is the microsecond and is defined in a 32-bit value. The granularity of the timestamp is limited by the capability of the device. The time base is reset to zero within the PDU\_IOCTL\_RESET function and after boot-up. All of the logical links, events and errors of one device derive their timestamps from the same time base.

The D-PDU API does not have any mechanism to detect a timestamp overflow. The application shall take care of an overflow.

#### 8.1.6.2 Timestamp for transmitted messages

For all UART-based protocols, the timestamp will be taken at the end (last bit) of the message. In the case of a controller-based protocol, the timestamp will be taken when the controller indicates the successful transmission of the message or the last frame of a message.

#### 8.1.6.3 Timestamp for received messages

For all UART-based protocols, the timestamp will be taken at the end (last bit) of the message. In the case of a controller-based protocol, the timestamp will be taken when the controller indicates the successful reception of the message or the last frame of a message. Indication of the start of a message or first frame is handled as an event and is described below.

#### 8.1.6.4 Timestamp for events, errors and indications

For events and errors, the timestamp will be taken when the event or error is detected.

#### 8.1.6.5 Timestamp for start of message indication

The indication of the start of message (see Table D.5) will be handled as outlined below.

- For UART-based protocols, the timestamp will be taken at the first bit of the received message. In this case, the timestamp shall be calculated (see example below).

EXAMPLE To calculate the timestamp for the first bit of a 9 600 baud line running 8N1, the equation would be:

$$\text{First bit time stamp} = \text{first byte time stamp} - \left[ \left( \frac{1 \text{ seconds}}{9\,600 \text{ bits}} \right) \times 10 \text{ bits} \right]$$

- For controller-based protocols, the timestamp will be taken when the controller indicates the reception of the first frame of the message.

## 8.2 API function overview and communication principles

### 8.2.1 Terms used within the D-PDU API

#### 8.2.1.1 Resource

A resource defines a communication channel towards a single ECU or towards multiple ECUs. It covers diagnostic protocols, and MVI protocol module hardware (transceivers, UART, multiplexer, etc.), including the Diagnostic Link Connector (DLC) of the MVI protocol module.

### 8.2.1.2 ComLogicalLink

A ComLogicalLink (CLL) defines a logical communication channel towards a single ECU or towards multiple ECUs. The configuration of the CLL is based on the selected bustype, pins and protocol. A CLL can be created on an existing and available resource. The protocol for the CLL is configured using ComParams. The D-PDU API is not restricted in the number of CLLs opened on a single resource, unless there are limitations for the protocol. The client application shall be aware of conflicts on resources.

### 8.2.1.3 ComPrimitive

A ComPrimitive (CoP) is a basic communication element holding data and controlling the exchange of data between the D-PDU API implementation and the ECU.

### 8.2.1.4 ComParam

A ComParam is a protocol communication parameter used to define the functionality of the vehicle communication protocol selected for a ComLogicalLink. Each protocol has a set of applicable ComParams that are set to a default value upon creation of a ComLogicalLink. A set of ComParams is used to individually define communication to a single ECU or a functional group of ECUs.

## 8.2.2 Function overview

Table 2 lists all D-PDU API functions and classifies them in functional groups.

**Table 2 — D-PDU API functions overview**

Function	Description
<b>General functions</b>	
PDUConstruct	Constructor with optional, manufacturer-specific arguments. Required to call for each D-PDU API implementation prior to any other D-PDU API function call.
PDUDeconstruct	Destructor required as last D-PDU API function call to free all resources.
PDUModuleConnect	Connect to a specific MVCI protocol module.
PDUModuleDisconnect	Disconnect from a specific MVCI protocol module.
PDURegisterEventCallback	Register or unregister a callback function for event notification.
PDUIoCtl	Invokes I/O control functions of an MVCI protocol module or ComLogicalLink.
<b>Information functions</b>	
PDUGetVersion	Obtains version information for a specified MVCI protocol module and its D-PDU API implementation.
PDUGetStatus	Obtains runtime information (e.g. status, life sign, etc.) from an MVCI protocol module, ComLogicalLink or ComPrimitive.
PDUGetLastError	Obtains the code for the last error that occurred in an MVCI protocol module or ComLogicalLink.
PDUGetTimestamp	Obtains the Timestamp information for a specific MVCI protocol module.
PDUGetObjectID	Obtains an ID for a given short name for a PDUObjectType. This is in addition to the possibility of parsing the MDF file.
<b>Resource management</b>	
PDUGetResourceIDs	Obtains all resource IDs that match the requested resource information.
PDUGetResourceStatus	Obtains the status of the requested resource ID.
PDUGetConflictingResources	Obtains a list of resource IDs that are in conflict with the given resource ID (e.g. two resources sharing the same physical controller).
PDUGetModuleIDs	Obtains the IDs of all MVCI protocol modules currently connected to the D-PDU API.
PDULockResource	Obtains a lock on the requested resource ID.
PDUUnlockResource	Releases the lock on the requested resource ID.

Table 2 — (continued)

Function	Description
<b>ComLogicalLink handling</b>	
PDUCreateComLogicalLink	Create a ComLogicalLink for a given D-PDUResource.
PDUDestroyComLogicalLink	Destroy a ComLogicalLink.
PDUConnect	Physically connects the previously created ComLogicalLink to the communication line.
PDUDisconnect	Physically disconnects the previously connected ComLogicalLink from the communication line.
<b>Link and ComParam handling</b>	
PDUGetComParam	Obtains current value of specified ComParam.
PDUSetComParam	Sets specified ComParam to given value. Overwrites previous values.
PDUGetUniqueRespIdTable	Obtains a table of Unique Response Identifiers. Each Unique Response Identifier is associated with a set of ComParams used to uniquely define an ECU response.
PDUSetUniqueRespIdTable	Set a table of Unique Response Identifiers. Each Unique Response Identifier value is assigned by the application.
<b>Message handling</b>	
PDUStartComPrimitive	Start the operation of given ComPrimitive (e.g. sending/receiving data).
PDUCancelComPrimitive	Cancel current execution of given ComPrimitive (e.g. cancel a running periodic send operation or an operation which has not yet been executed).
PDUGetEventItem	Retrieve item data (e.g. received data) for given event source (i.e. MSCI protocol module, ComLogicalLink and ComPrimitive).
PDUDestroyItem	Destroy given item.

### 8.2.3 General usage

The sequences of function calls differ between the D-PDU API and existing APIs like for SAE J2534-1 and RP1210a. Therefore, the following subclauses show a minimal sequence of calls in order to facilitate understanding. Also, the terms “asynchronous” and “synchronous” communication are defined in the subsequent subclauses.

In order to initialize the D-PDU API implementation and to prepare communication for one channel, the application needs to go through the following minimal sequence of API function calls.

Table 3 lists the D-PDU API functions in sequential order to facilitate better understanding.

Table 3 — General usage of D-PDU API function calls

Action #	Function call from application	D-PDU API
<b>Initial connection to D-PDU API library</b>		
1	PDUConstruct  PDUGetModuleIds  PDURegisterEventCallback	Initialize D-PDU API library. This does not necessarily connect to an MVCI protocol module. This begins the process of figuring out what MVCI protocol modules are available. Information about the available modules is retrieved via the PDUGetModuleIds function.  Retrieve the list of available MVCI protocol modules and their handles and modules types.  (optional) Register global system callback.
<b>Initial connection to an MVCI protocol module</b>		
2	PDUModuleConnect  PDURegisterEventCallback	Connect the D-PDU API library to one or more MVCI protocol modules.  (optional) Register module callback.
<b>Setting up a ComLogicalLink</b>		
3	PDUCreateComLogicalLink  PDURegisterEventCallback  PDUGetUniqueRespIdTable  PDUGetComParam  PDUSetComParam  PDUSetUniqueRespIdTable  PDUConnect	Create a ComLogicalLink (based on protocol, pins and Bustype).  (optional) Register ComLogicalLink callback.  Retrieve the applicable set of ComParams for unique identification of ECUs determined by the specific protocol for the ComLogicalLink.  Get ComParam item structures initialized with the default ComParams for the selected protocol.  Set the ComParams for the ComLogicalLink.  Configure the Unique Response Identifier Table for all possible ECU Responses to be received on the ComLogicalLink.  Connect the ComLogicalLink to the communication line.
<b>Starting vehicle communications on a ComLogicalLink</b>		
4	PDUStartComPrimitive (PDU_COPT_STARTCOMM or PDU_COPT_SENDRECV)	A PDU_COPT_STARTCOMM ComPrimitive is required for certain protocols as the first ComPrimitive (e.g. ISO 14230). A PDU_COPT_STARTCOMM ComPrimitive is also used to control the ability to start sending tester present messages (see CP_TesterPresentHandling). Once tester present handling is enabled, the message is sent immediately, prior to the initial tester present cyclic time (CP_TesterPresentTime).  For all other vehicle bus communications, a PDU_COPT_SENDRECV ComPrimitive is used to begin vehicle communications.
<b>Using a ComLogicalLink for vehicle communications</b>		
5	PDUStartComPrimitive (PDU_COPT_SENDRECV)  PDUSetComParam  PDUGetEventItem  PDUDestroyItem	Set up and start ComPrimitives for vehicle bus activity: Send only, Send Receive, or Receive only.  Change ComParams (temporary changes per ComPrimitive or permanent changes per PDU_COPT_UPDATEPARAM type of ComPrimitive).  Retrieve event items.  Destroy event item memory from D-PDU API memory.
<b>Stopping vehicle communications on a ComLogicalLink</b>		
6	PDUStartComPrimitive (PDU_COPT_STOPCOMM)	A PDU_COPT_STOPCOMM ComPrimitive will stop all communication on a ComLogicalLink.

Table 3 — (continued)

Action #	Function call from application	D-PDU API
<b>Connecting to a newly available MVCI protocol module</b>		
7	<p>Receive callback indicating new MVCI protocol module detected by D-PDU API.</p> <p>PDUGetModuleIds</p> <p>PDUModuleConnect</p> <p>PDURegisterEventCallback</p>	<p>Receive a system event callback with an information type PDU_IT_INFO indicating the list of modules has changed: PDU_INFO_MODULE_LIST_CHG</p> <p>Retrieve the new list of available MVCI protocol modules and their handles and modules types. Any previously detected MVCI protocol modules will return the same hMod handles.</p> <p>NOTE If detection of an MVCI protocol module was lost and then detection is re-established, the module handle (hMod) will not be the same as the previous handle. This ensures that any event items stored in the initial connection will still be available for reading prior to a PDUModuleDisconnect.</p> <p>Connect to the new MVCI protocol module.</p> <p>(optional) Register module callback.</p>
<b>Loss of communication to an MVCI protocol module and reconnection</b>		
8	<p>Receive callback indicating communications to an MVCI protocol module has been lost.</p> <p>PDUModuleDisconnect</p> <p>Wait for a Callback indicating status change of MVCI protocol module.</p> <p>PDUGetModuleIds</p>	<p>Receive a module event callback indicating communication is lost to an MVCI protocol module. (See PDU_ERR_EVT_LOST_COMM_TO_VCI). The hMod information is part of the callback. The client application should call PDUModuleDisconnect after all items have been retrieved from the module event queue.</p> <p>NOTE 1 All ComPrimitives active for all ComLogicalLinks on the Module will generate a PDU_COPST_CANCELLED status event item. All ComLogicalLinks on the Module will generate a PDU_CLLST_OFFLINE status event item.</p> <p>NOTE 2 If the MVCI protocol module still maintains power after it has lost communication with a client session, it disconnects from all vehicle buses associated with that client session and self cleans up all resources associated with the client session.</p> <p>NOTE 3 The module handle (hMod) is preserved until a PDUModuleDisconnect is called.</p> <p>Once PDUModuleDisconnect is called, the hMod handle is no longer valid, all event items in the queues are freed and any related module memory reserved by the D-PDU API library is unreserved.</p> <p>Receive a system event callback with an information type PDU_IT_INFO indicating the list of modules has changed: PDU_INFO_MODULE_LIST_CHG.</p> <p>Retrieve the new list of available MVCI protocol modules and their handles and modules types. Observe that the hMod from the MVCI Protocol module that had a communication loss is no longer listed as an available hMod.</p> <p>NOTE 4 See Step 7 (above) for steps to reconnect to a module after a loss of communications event.</p>
<b>Disconnect from a MVCI protocol module</b>		
9	<p>PDUDisconnect</p> <p>PDUDestroyComLogicalLink</p> <p>PDURegisterEventCallback(NULL)</p> <p>PDUModuleDisconnect</p>	<p>Disconnect the ComLogicalLink from the communication line.</p> <p>Destroy the ComLogicalLink.</p> <p>(optional) Unregister the module event callback. From now on no events will be signalled to the application for the module.</p> <p>Disconnect a specific MVCI protocol module from the D-PDU API library and free all reserved memory resources.</p>

Table 3 — (continued)

Action #	Function call from application	D-PDU API
<b>Disconnection from the D-PDU API library</b>		
10	PDUDeconstruct	De-initialize the D-PDU API and destroy/free any internal resources.

## 8.2.4 Asynchronous and synchronous communication

### 8.2.4.1 General

The asynchronous communication operation of the D-PDU API implies that calls to the API are immediately returned even though the requested activity might still be running or is still waiting for execution inside the D-PDU API implementation. The D-PDU API uses asynchronous calls to support all types of vehicle communication requirements (e.g. non-cyclic, cyclic, event driven communication, etc.), as well as status changes and error detection events.

The synchronous communication operation of the D-PDU API implies that calls to the API are returned with the requested information to the application (e.g. PDUGetComParam).

The D-PDU API functions allow the application to use both event callback (asynchronous) and polled (synchronous) communication principles to exchange data with the D-PDU API.

### 8.2.4.2 Event callback (asynchronous mode)

In this case, the communication between application and the D-PDU API implementation is completely event driven. The application may register an application-specific event callback function by calling PDURegisterEventCallback. Any events queued into an empty ComLogicalLink Event queue or the events that are already queued at the point in time the callback function is registered will cause the callback function to be invoked. The callback function will be called on the thread of the D-PDU API. It is the responsibility of the application to minimize the time spent in this callback. This specification suggests that the application callback function post an event to wake another thread to do the reading of the event data. If the application shall make a D-PDU API function call in the callback routine, then PDUGetEventItem is the only permitted call.

### 8.2.4.3 Polling (synchronous mode)

In this case, the application does not make use of the event callback mechanism. The application initiates the D-PDU API functions (just as in asynchronous mode) and uses the PDUGetStatus and PDUGetEventItem functions to detect status changes and to read event items from the event queues.

## 8.2.5 Usage of resource locking and resource unlocking

A ComLogicalLink has the ability to lock different elements of a physical resource it is connected to using the D-PDU API function PDUlockResource. Through locking, a ComLogicalLink can prohibit all other ComLogicalLinks access to the resource if necessary. For example, if there are two ComLogicalLinks sharing a vehicle bus, and an ECU needs to be reprogrammed, the physical resource (the vehicle bus) can be locked during the reprogramming sequence. The API function PDUUnlockResource is used to release the lock on a physical resource.

## 8.2.6 Usage of ComPrimitives

### 8.2.6.1 ComPrimitive overview

To provide a generic data exchange mechanism for different communication principles, several ComPrimitive types are specified. The behaviour and usage of each ComPrimitive type depends on the specific communication protocol used with a ComLogicalLink. These issues have to be described for each specific communication protocol implementation.

Table 4 describes the different ComPrimitive types.

**Table 4 — ComPrimitives overview**

ComPrimitive type	Description
PDU_COPT_STARTCOMM	Start communication with ECU by sending an optional request. The detailed behaviour is protocol dependent. For certain protocols (e.g. ISO 14230), this ComPrimitive is required as the first ComPrimitive. This ComPrimitive is also required to put the ComLogicalLink into the state PDU_CLLST_COMM_STARTED which allows for tester present messages to be enabled (see CP_TesterPresentHandling). Once tester present handling is enabled, the message is sent immediately, prior to the initial tester present cyclic time (CP_TesterPresentTime).
PDU_COPT_STOPCOMM	Stop communication with ECU by sending an optional request. The detailed behaviour is protocol dependent. After successful completion of this ComPrimitive type, the ComLogicalLink is placed into PDU_CLLST_ONLINE state and no further tester presents will be sent. A PDU_COPT_STARTCOMM ComPrimitive might be required by some protocols (e.g. ISO 14230) to begin communications again.
PDU_COPT_UPDATEPARAM	Copies ComParams and the UniqueRespIdTable related to a ComLogicalLink from the working buffer to the active buffer. Prior to update, the values need to be passed to the D-PDU API by calling PDUSetComParam and/or PDUSetUniqueRespIdTable, which modifies the working buffer. NOTE 1 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time. NOTE 2 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func, CP_P3Phys.
PDU_COPT_SENDFUNC	Send request data and/or receive corresponding response data (single or multiple responses). See 10.1.4.17 for detailed settings.
PDU_COPT_DELAY	Wait the given time span before executing the next ComPrimitive.
PDU_COPT_RESTORE_PARAM	Converse functionality of PDU_COPT_UPDATEPARAM. Copies ComParams related to a ComLogicalLink from active buffer to working buffer.

**8.2.6.2 ComPrimitive send/receive cycle overview**

**8.2.6.2.1 General**

Each ComPrimitive is controlled by a PDU\_COP\_CTRL\_DATA structure (see 10.1.4.17). Table 5 describes how the send and receive cycles are used. For more examples, see also Figures 8 to 16.

**Table 5 — ComPrimitives send/receive cycles**

PDU_COPT_SENDFUNC, PDU_COPT_STARTCOMM, PDU_COPT_STOPCOMM		
	NumSendCycles	NumReceiveCycles
SEND and RECEIVE	Number of send cycles to be performed; -1 for infinite send operation.	Number of expected responses per request; -1 for infinite receive operation, -2 for multiple responses.
SEND ONLY	Number of send cycles to be performed; -1 for infinite send operation	0
RECEIVE ONLY	0	Number of receive cycles to be performed; -1 for infinite receive operation, -2 for multiple responses.
No data to transmit or receive (e.g. PDU_COPT_STARTCOMM with no pPduData message)	0	0

### 8.2.6.2.2 NumReceiveCycles description

The NumReceiveCycles is the number of expected complete responses for a ComPrimitive. An infinite receive ComPrimitive (-1) will usually never finish (i.e. never generate a PDU\_COPST\_FINISHED status item, see PDU\_STATUS\_DATA) unless timing is enabled for cyclic responses (see ComParam CP\_CyclicRespTimeout) and therefore shall usually be cancelled (see 8.4.18).

### 8.2.6.2.3 NumSendCycles description

The NumSendCycles is the number of periodic transmits to be sent on the vehicle bus. The periodic time interval is specified in milliseconds in the PDU\_COP\_CTRL\_DATA structure (see 10.1.4.17). If NumSendCycles is equal to -1, it is considered an infinite send ComPrimitive. An infinite send ComPrimitive will never finish and shall always be cancelled (see 8.4.18). A periodic send ComPrimitive will have state transitions to and from PDU\_COPST\_EXECUTING to PDU\_COPST\_WAITING for each periodic interval.

Table 6 defines some possible combinations of ComPrimitive types and cycle types.

**Table 6 — ComPrimitives combinations**

NumSendCycles	NumReceiveCycles	Description
1	1	Send one request and look for one response. If a response is not received before a receive timeout occurs (e.g. P2Max), then the ComPrimitive will complete and set a receive timeout error event.
-1	1	Continuously send requests and look for one response per request. If the response is not received before a receive timeout occurs (e.g. P2Max), the ComPrimitive will set a receive timeout error event (PDU_ERR_EVT_RX_TIMEOUT) and proceed to the next send interval.
1	3	Send one request and look for three responses. The ComPrimitive is completed when three responses are received or a receive timeout occurs (e.g. P2Max).
1	-1	Send one request and look for continuous responses. This case is equivalent to ODX IS-CYCLIC = TRUE. The application shall cancel this ComPrimitive since it will never finish. A cyclic receive timeout can be used if the NumReceiveCycles is set to infinite (-1). (See CP_CyclicRespTimeout ComParam.) In the case of a cyclic receive timeout, the ComPrimitive will transition to PDU_COPST_FINISHED.
1	-2	Send one request and wait for a receive timeout (e.g. P2Max). Should be used for functional addressing when number of responses is unknown or with physical addressing with possible unknown number of responses. This case would be appropriate for ODX IS-MULTIPLE = TRUE.
1	0	Send one message and ignore any responses.
-1	0	Continuously send messages and ignore any responses.
0	-1	Continuously look for received messages. A cyclic receive timeout will be used if the NumReceiveCycles is set to infinite (-1). (See CP_CyclicRespTimeout ComParam.)
0	>0	Look for the specified number of received messages. A cyclic receive timeout will be used if the NumReceiveCycles is set to >0. (See CP_CyclicRespTimeout ComParam.)

**NOTE** If cyclic time is set to 0 in the PDU\_COP\_CTRL\_DATA, then the ComPrimitive is put on the transmit queue after each completion cycle, but is at a lower priority than other ComPrimitives and Tester Present Messages.

### 8.2.6.3 ComPrimitive principles

#### 8.2.6.3.1 General

The following subclauses describe how to use ComPrimitives for different communication principles known from automotive communication protocols. All of the described actions assume the D-PDU API has been initialized and a ComLogicalLink has been created (for details, see Table 3). In addition to the D-PDU API function calls shown in

the tables, additional API calls can be used for additional functions (e.g. status requests, etc.). Any memory allocation or de-allocation initiated by create, start and destroy calls is handled within the D-PDU API.

### 8.2.6.3.2 Starting communication

To initiate communication between a tester and an ECU, different initialization methods exist for various communication protocols. For OBD initialization handling, see Annex J. The list below describes different standard initialization use cases.

#### a) No initialization (using PDU\_COPT\_SENDDRECV)

- Directly start sending a request to the ECU using a PDU\_COPT\_SENDDRECV ComPrimitive. This method is used for protocols like CAN and SAE J1850 which do not require an initialization sequence.

#### b) No initialization (using PDU\_COPT\_STARTCOMM)

- Directly start sending a request to the ECU after a PDU\_COPT\_STARTCOMM ComPrimitive. This method is used for protocols like CAN and SAE J1850 which do not require an initialization sequence, but might use a ComPrimitive message to enter diagnostic mode (e.g. StartDiagnosticSession service). An optional message can be sent for the PDU\_COPT\_STARTCOMM ComPrimitive for these protocols.
- No initialization response: If an optional message is sent with NumReceiveCycles != 0, then the D-PDU API waits for a response message. If an optional message is not sent, the ComPrimitive finishes right away and no result data is returned. The internal state of the ComLogicalLink changes accordingly (PDU\_CLLST\_COMM\_STARTED).

#### c) 5-baud initialization (using PDU\_COPT\_STARTCOMM)

- Initialize the communication link to the ECU by sending a 5-baud initialization sequence. This method is used for many K-Line protocols, e.g. KWP2000. During 5-baud initialization, the tester sends the ECU address on the bus and then calculates the baud rate to be used for further communication. See ComParams (CP\_InitializationSettings, CP\_5BaudMode, CP\_5BaudAddressFunc, CP\_5BaudAddressPhys).
- 5 baud initialization response The NumReceiveCycles shall be set to 1 for the ECU key bytes to be returned to the client application, in this case, a result message data structure (see 10.1.4.11.4 PDU\_RESULT\_DATA) will contain the key bytes in the following order:
  - PDU[0]=key byte 1;
  - PDU[1]=key byte 2.
- A PDUGetComParam can be called to read the CP\_Baudrate ComParam which will contain the calculated baud rate.
- No optional message will be allowed for a 5-baud initialization start communication request. The client application should send a new SendRecv ComPrimitive for communication after a 5-baud initialization completes.
- After a 5-baud initialization sequence completes, the protocol handler will begin sending keep-alive messages (see CP\_TesterPresent ComParams) if enabled.
- If the ECU key bytes indicate that extended timing is supported, the ComParam CP\_ExtendedTiming can be used to override the default values of ISO 14230-2.

#### d) Fast initialization (using PDU\_COPT\_STARTCOMM)

- Initialize the communication link to the ECU by sending a wake-up pattern optionally followed by a service request provided in the ComPrimitive request data.

- Fast initialization response: The PDU\_COPT\_STARTCOMM ComPrimitive behaves like a normal SendRecv ComPrimitive if an optional request message is contained in the data. Therefore, any ECU responses that match the expected response structure will be returned to the client application.
- If the ECU key bytes indicate that extended timing is supported, the ComParam CP\_ExtendedTiming can be used to override the default values of ISO 14230-2.

e) Tester Present messages (ComLogicalLink State = PDU\_CLLST\_COMM\_STARTED)

- Tester Present messages will only be enabled when the ComLogicalLink is in the state PDU\_CLLST\_COMM\_STARTED. A successful PDU\_COPT\_STARTCOMM ComPrimitive is required to enter this state (pCoPData for this ComPrimitive is optional). See CP\_TesterPresentHandling for more information on tester present message enabling. Once tester present handling is enabled, the message is sent immediately, prior to the initial tester present cyclic time (CP\_TesterPresentTime).

See Table 7 for a generic approach to starting communications on a ComLogicalLink.

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

Table 7 — Starting communications

Sequence	Action	Description
1	PDUCreateCommLogicalLink	Create a logical link on a physical resource. The state is PDU_CLLST_OFFLINE until connected.
2	PDUSetComParam	Set ComParams required for ECU communication, like ECU target address, initialization settings, etc.
3	PDUSetUniqueRespIdTable	Set of ComParams required for uniquely identifying different ECUs. A unique identifier that will be provided by the application during this function call is returned to the client application indicating which ECU response matched the ComPrimitive. The unique identifier is returned to the client application in a PDU_RESULT_DATA structure.
4	PDUConnect	Physically connect the resource to the vehicle bus. The state of the ComLogicalLink is now PDU_CLLST_ONLINE. The vehicle bus can now be monitored with receive only type of ComPrimitives. A transmit on the vehicle bus is possible via a PDU_COPT_STARTCOMM or PDU_COPT_SENDRECV ComPrimitive.
5	Optional: PDUStartComPrimitive (PDU_COPT_STARTCOMM)	<p>This ComPrimitive is placed on the ComPrimitive queue. Upon execution, the ECU communication will be initialized using the configured initialization method. An optional Start Communication Message can be sent to the ECU. ComParams are used to define the type of initialization to perform and the addressing type (physical/functional). In case of fast initialization, the request message, which is defined in the message data structure (pCopData), will be sent after the wakeup pattern.</p> <p>Immediately after successful initialization, the following occurs.</p> <p>Any ECU responses matching the CoP expected response structure are returned as result items (PDU_IT_RESULT).</p> <p>If tester present handling is enabled (see CP_TesterPresentHandling), the message is sent immediately, prior to the initial tester present cyclic time (CP_TesterPresentTime). After initial transmission, the periodic intervals are started.</p> <p>The ComLogicalLink is set to PDU_CLLST_COMM_STARTED.</p> <p>The ComPrimitive status is set to PDU_COPST_FINISHED.</p>
6	PDUStartComPrimitive (PDU_COPT_SENDRECV)	<p>Continue with ECU communication using ComPrimitives.</p> <p>NOTE Tester Present messages can only be enabled if the ComLogicalLink is in the state PDU_CLLST_COMM_STARTED (see sequence step 5 above).</p>

### 8.2.6.3.3 Stopping communication

The recommended approach for stopping communication can be found in Table 8.

Table 8 — Stopping communication

Sequence	Action	Description
1	....	Running ECU communication using ComPrimitives
2	PDUStartComPrimitive (PDU_COPT_STOPCOMM)	<p>This ComPrimitive is placed on the ComPrimitive queue. Upon execution, all ECU communication will be terminated. An optional Stop Communication Message can be sent to the ECU.</p> <p>If no message is sent to the ECU, the ComPrimitive status directly changes to PDU_COPST_FINISHED.</p> <p>Immediately after successful stopping of ECU communications, the following occurs.</p> <p>Any ECU responses matching the CoP expected response structure are returned as result items (PDU_IT_RESULT).</p> <p>The ComLogicalLink is set to PDU_CLLST_ONLINE.</p> <p>All currently queued ComPrimitives and currently executing ComPrimitives are cancelled (PDU_COPST_CANCELLED).</p> <p>Periodic Tester Present Messages are stopped.</p>

### 8.2.6.3.4 Send and receive handling

The recommended approach for send and receive message handling can be found in Table 9.

**Table 9 — ComPrimitive send/receive handling**

Function	Description
Sending Message Handling	<p>A ComLogicalLink shall be in the state PDU_CLLST_COMM_STARTED for Tester Present messages to be periodically sent. A ComLogicalLink can only be placed in a Comm Started state after a successful completion of a PDU_COPT_STARTCOMM ComPrimitive (see 8.2.6.3.2).</p> <p>When a protocol running on a ComLogicalLink has been properly configured using the ComParams for the selected protocol type, then the D-PDU API can correctly construct a full D-PDU to be sent to the vehicle's ECU. Proper construction of a message is based on raw mode configuration of the ComLogicalLink (see D.2.3) and is protocol specific. Therefore, proper construction could consist of message header bytes, transport layer handling and checksumming features. Furthermore, messages are constructed based on the type of addressing scheme selected (physical or functional addressing defined as a ComParam).</p> <p>In non-raw mode, the first byte of the pCopData of the ComPrimitive received from the application would usually consist of the service ID (if applicable to the protocol).</p>
Receive Message Handling	<p>Every message received from the vehicle bus will first be subjected to the Pass/Block filters (if configured). Initially, the D-PDU API configures the filters based on the Unique Response Identifier Table. The client application can override this auto configuration of the filters by using any of the PDUIoctl commands PDU_IOCTL_XXX_MSG_FILTER.</p> <p>Next, the ECU message will be checked for correctness (checksum, PCI information, etc).</p> <p>The UniqueRespIdTable is then referenced to determine a match to a known ECU identifier. The UniqueRespIdentifier table can be configured to pass all ECU identification (see 8.4.28.7). The UniqueRespIdentifier will be used in the PDU_IT_RESULT event item to indicate to the application which ECU the message belongs to.</p> <p>Finally, the D-PDU API will determine a match of the ECU message to a ComPrimitive Expected Response Structure (see 10.1.4.19). The D-PDU API will first compare the ECU message to the current active Send/Receive ComPrimitive and if not matched it will search through the Receive Only list. The message is considered bound when it matches to the first ComPrimitive and no further ComPrimitive matching is continued after the initial match.</p> <p>If the response cannot be bound to any ComPrimitive's ExpectedResponseStructure, the message is discarded.</p> <p>NOTE 1. A transport layer uses the UniqueRespIdentifier table and the ComParams from the currently active SendRecv ComPrimitive for initial receive handling of frames/messages. If the ComLogicalLink does not have an active SendRecv ComPrimitive, then the ComLogicalLinks active ComParam buffer is used. Once the frame/message is bound to a ComPrimitive, the set of ComParams attached to the ComPrimitive is used for any further processing (e.g. receive timing).</p>
SEND ONLY  NumReceiveCycles = 0 (PDU_COPT_SENDRECV)	<p>If the NumReceiveCycles is equal to zero and the NumSendCycles is not equal to zero, then the ComPrimitive is considered to be a send only ComPrimitive. This type of ComPrimitive can still be periodic by setting the delay time interval in the PDU_COP_CTRL_DATA structure (see 10.1.4.17).</p> <p>The D-PDU API will construct the proper message to be transmitted, but will not set up any receive timers for responses from an ECU(s). Any ComParams defined for "time between transmits" (i.e. CP_P3min, CP_P3Func, CP_P3Phys) will be used to ensure proper vehicle bus timing.</p> <p>If the protocol type allows for an immediate transmit of another message, then any pending ComPrimitives would be available for immediate execution.</p> <p>Once the ComPrimitive has completed all transmission send cycles (NumSendCycles), the status of the ComPrimitive is set to PDU_COPST_FINISHED and the status item (see 10.1.4.11.1) is placed on the ComLogicalLink's Event Queue.</p>

Table 9 — (continued)

Function	Description
RECEIVE ONLY  NumSendCycles = 0 (PDU_COPT_SENDRXCV)	<p>If the NumSendCycles is equal to zero and the NumReceiveCycles is not equal to zero, then the ComPrimitive is considered to be a receive only ComPrimitive. The NumReceiveCycle count is used to monitor the vehicle bus for messages that match the ExpectedResponseStructure (see 10.1.4.19). When the receive count has been reached, the ComPrimitive transitions to PDU_COPST_FINISHED.</p> <p>Once all the expected responses have been received, the status of the ComPrimitive is set to PDU_COPST_FINISHED and the status item (see Structure for status) is placed on the ComLogicalLink's Event Queue.</p> <p>NOTE 2 A cyclic receive timeout can be used if the NumReceiveCycles is set to infinite (-1) or &gt;0 (see CP_CyclicRespTimeout ComParam). In the case of a cyclic receive timeout, the ComPrimitive will transition to PDU_COPST_FINISHED. Otherwise, the application cancels the ComPrimitive via PDUCancelComPrimitive.</p> <p>NOTE 3 No pCopData bytes are supplied in this ComPrimitive type.</p>
SEND AND RECEIVE  NumReceiveCycles != 0 NumSendCycles != 0 (PDU_COPT_SENDRXCV)	<p>When both the NumSendCycles and the NumReceiveCycles are not equal to zero, the ComPrimitive will attempt to deliver the number of responses specified in NumReceiveCycles each time the ComPrimitive is sent. If the number of responses specified by NumReceiveCycles is not found before a receive timeout occurs (e.g. P2Max), the ComPrimitive will generate an error event, indicating that a receive timeout occurred. If the ComPrimitive is periodic, it will NOT transition to PDU_COPST_FINISHED even on a receive timeout, the cyclic transmissions will continue.</p> <p>The D-PDU API will construct the proper message to be transmitted. Once the message has been properly transmitted, the receive timers for the ComLogicalLink will be enabled waiting for matching ECU responses. See ExpectedResponseStructure and PDU_COP_CTRL_DATA (see 10.1.4.17).</p> <p>Multiple Expected Responses (IS-MULTIPLE) (see 10.1.4.17, PDU_COP_CTRL_DATA - NumReceiveCycles = -2): Each received message will reset the receive timer (e.g. P2Max) on a matching response. A receive timeout with no matching responses from any ECU will generate an error event (PDU_ERR_EVT_RX_TIMEOUT). The ComPrimitive will end without an error when a receive timeout has occurred (e.g. P2Max) and at least one valid response has been received. All functional requests with expected responses should set the NumReceiveCycles to IS-MULTIPLE (-2).</p> <p>Infinite Responses (IS-CYCLIC) (see 10.1.4.17, PDU_COP_CTRL_DATA - NumReceiveCycles = -1): When the ComLogicalLink completes the transmit and receives the first positive response, the ComPrimitive is placed into a receive only mode. This allows other ComPrimitives to be transmitted on the ComLogicalLink while still receiving responses from the cyclic ComPrimitive. A cyclic receive timeout can be used if the NumReceiveCycles is set to infinite (-1). (See CP_CyclicRespTimeout ComParam). In the case of a cyclic receive timeout, the ComPrimitive will transition to PDU_COPST_FINISHED. Otherwise, the application shall cancel the ComPrimitive via PDUCancelComPrimitive.</p> <p>Once all the expected responses have been received and all transmission send cycles have been completed, the status of the ComPrimitive is set to PDU_COPST_FINISHED and the status item (see 10.1.4.11.1) is placed on the ComLogicalLink's event queue.</p>

Table 9 — (continued)

Function	Description
7F Handling	<p>This level of protocol message handling has been moved to the D-PDU API to ensure proper low-level real-time requirements. This functionality can be handled in RawMode if the Response Code Offset ComParam is correctly configured and the number of header bytes can be determined by the protocol handler (some protocols allow configuration of header byte count via ComParams).</p> <p>Negative response (0x7F) handling can be enabled or disabled by properly setting the correct ComParams in a ComLogicalLink. (See ComParam CP_RCxxHandling in Annex B.)</p> <p>Only response codes 0x21, 0x23 and 0x78 are configurable to be handled by the D-PDU API. Not all protocol implementations support all negative responses codes.</p> <p>The ComLogicalLink shall actively process a ComPrimitive with expected responses to a request to proceed with any 0x7F handling (i.e. negative response handling is not enabled for receive only ComPrimitives).</p> <p>The D-PDU API will bypass any addressing information (based on protocol) to determine if an ECU has responded with a 0x7F. The 0x7F code is typically contained in the first byte of the payload data, followed by the requested service id and the Response Code (see CP_RCByteOffset for configurable response code byte offset processing). The Response Code is extracted from the payload data and used to determine if further 0x7F handling is necessary by the D-PDU API.</p> <p>Based on the type of Response Code, the D-PDU API will begin the timing/retry handling. The D-PDU API is protected from infinite retries by CP_RCxx CompletionTimeout ComParam. If the ECU fails to generate a positive response in the specified Completion Timeout time, then the D-PDU API will generate an error event item indicating no responses received PDU_ERR_EVT_RX_TIMEOUT.</p> <p>If negative response handling is not enabled or the response code is not recognized by the D-PDU API, then the negative response message is placed in a result item and added to the ComLogicalLink's Event Queue.</p>

NOTE PDU\_ERR\_INVALID\_PARAMETERS will be returned for ComPrimitives of type PDU\_COPT\_SENDRXCV when NumSendCycles>0 and PDU is empty. When the NumSendCycles is set =0, the PDU will be ignored.

#### 8.2.6.3.5 ComPrimitives in non-raw mode

The idea of non-raw mode is that everything about an ECU or a functional group of ECUs is known and can be configured using ComParams for the selected protocol. The ComParam information would normally be contained in a database. A database schema, such as ODX, and a COMPARAM-SPEC have been developed to support the concept of fully abstracting ComParams to be used by the D-PDU API.

Each ComLogicalLink is directed to a single ECU and/or a functional group of ECUs. The ECU information can be configured initially and not changed during the course of ECU communication. If switching between ECUs is necessary on a single vehicle serial bus, then it is suggested that the application create another ComLogicalLink for the additional ECU communication.

It shall be noted that an additional ComLogicalLink cannot be created if it would result in the use of a non-shareable resource (e.g. DLC pins have already been reserved by another ComLogicalLink with a different protocol id). Also, if a second ComLogicalLink requires a "Start Communication Message" which causes the vehicle bus pins to be driven into a different impedance state, this situation would have an undeterminable behaviour on the vehicle bus. All other ComLogicalLinks that are sharing the resource would begin to report errors because communication has been lost to their specific ECU (e.g. ISO 14230/ISO 9141-2 initialization sequence would cause another ComLogicalLink using the same protocol and pins to lose communication to an ECU).

#### 8.2.6.3.6 ComPrimitives in raw mode (PassThru)

The idea of raw mode is that everything about an ECU or a functional group of ECUs and the protocol is known by the client application. The application takes on the responsibility over the entire protocol message structure, including header bytes and checksums. The only exception would be the necessary requirements of things like the transport layer (see ISO 15765-2) and inter-byte/inter-message timing (see ISO 9141-2/ISO 14230).

The application shall handle all knowledge about protocol header byte configuration, CAN IDs, extended addressing, negative responses, receive timeout timing (e.g. P2Max), tester present handling and the binding of response(s) to the request. Therefore, the application cannot be protocol-independent (abstract) in its use of the D-PDU API.

For ISO 15765, SAE J1939 and ISO 11898, the first 4 bytes of the pCopData shall be the CAN ID (11-bit or 29-bit). If extended addressing is enabled (see D.2.1), then the byte following the CAN ID contains the extended address byte.

8.2.6.4 ComPrimitive sequence diagrams

8.2.6.4.1 Single request/single response (master/slave communication) — Event notification

This communication principle is used for classic master/slave communication (e.g. ISO 14230 KWP 2000 standard services). See Figure 8.

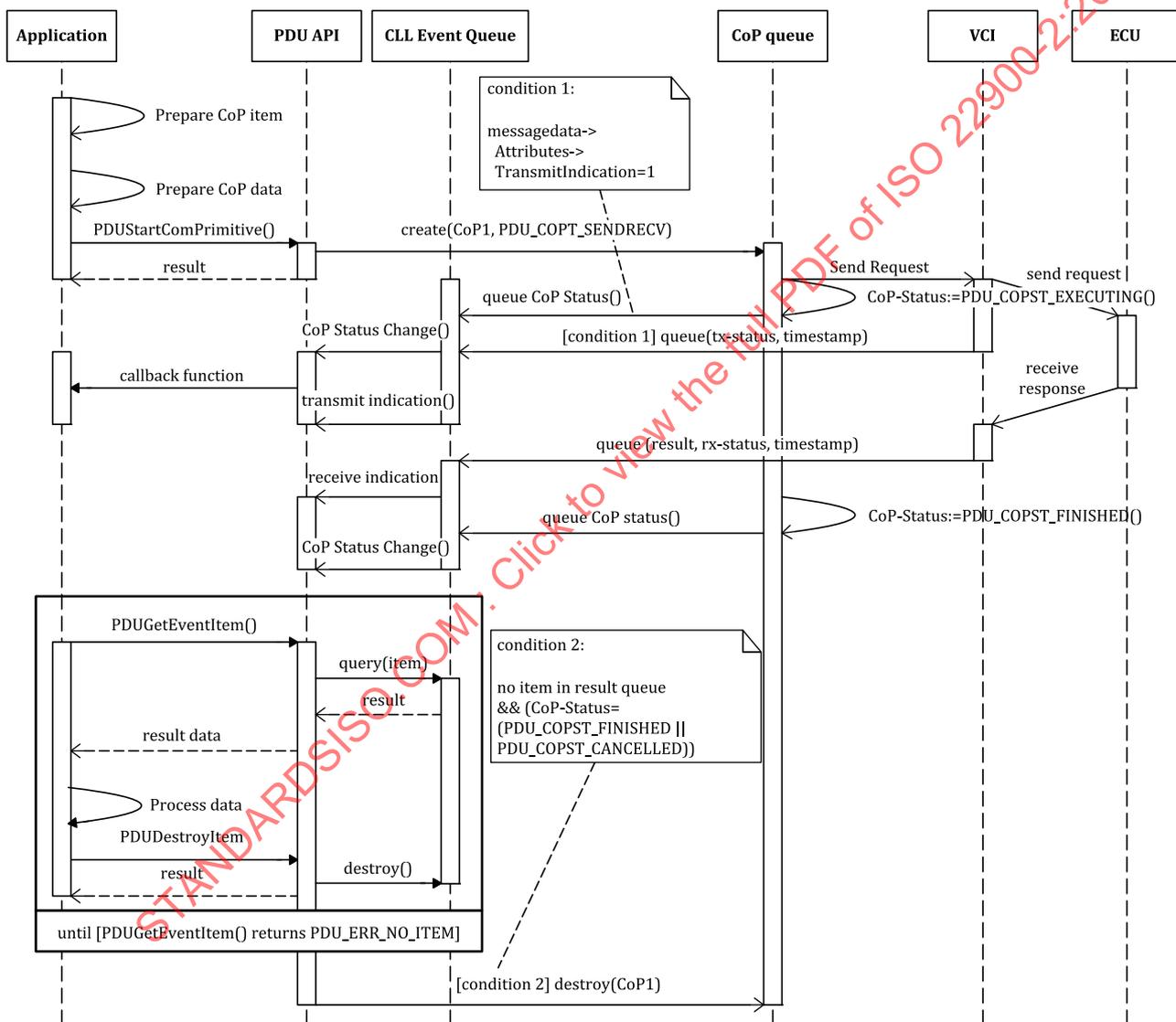


Figure 8 — Single request-single response (master, slave communication)

NOTE A callback is only initiated when an event is placed into an empty queue.

8.2.6.4.2 Single request/single response (master/slave communication) — Polling mode

Applications working without event notification can poll for results related to a specific ComPrimitive by calling `PDUGetStatus`. In this case, the callback function will not be called as depicted in Figure 9.

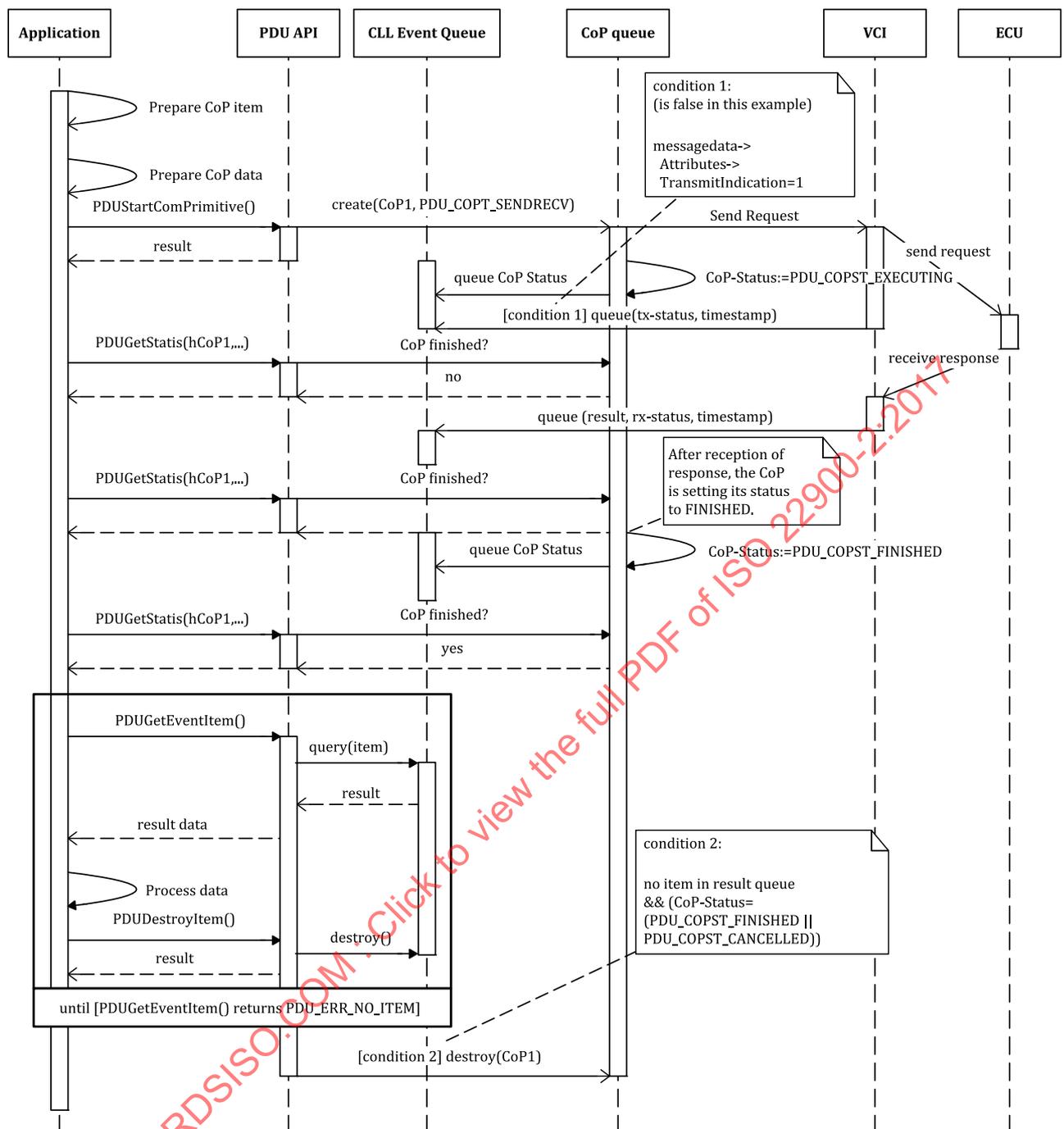


Figure 9 — Single request-single response (master, slave communication) — Polling mode

### 8.2.6.4.3 Single request/multiple responses

This communication principle is used in many protocols for finite and infinite communication sequences. The basic principle for this ComPrimitive type is shown in Figure 10.

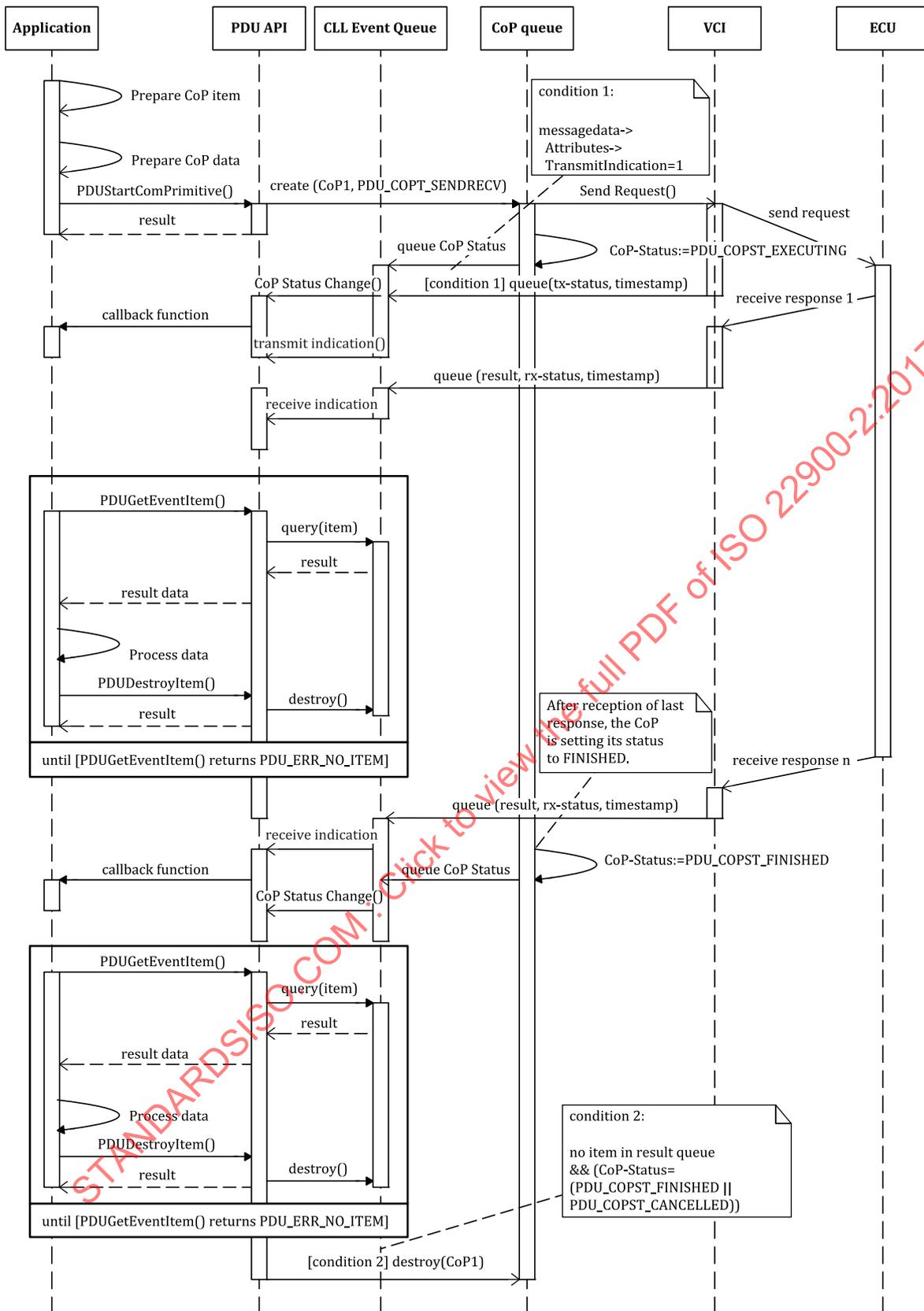


Figure 10 — Single request, multiple responses

NOTE The API function calls `PDUGetEventItem` and `PDUDestroyItem`, as well as the result data processing by the application, can be executed while the D-PDU API's receiving process of ECU responses runs in parallel.

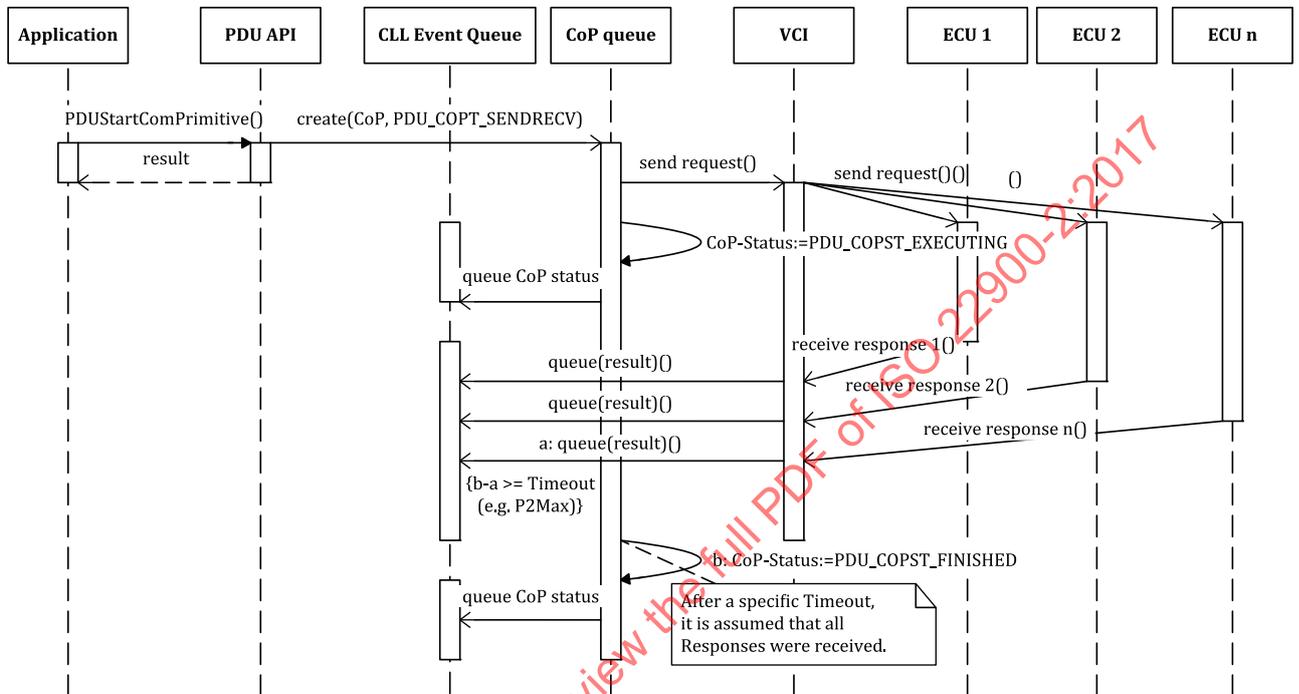
An event callback does not have to be called for each data item. It is up to the callback routine to check for further event items before returning. A callback is initiated either when an event is placed into an empty queue or when

events are already queued when the PDURegisterEventCallback function is called that registers a callback function.

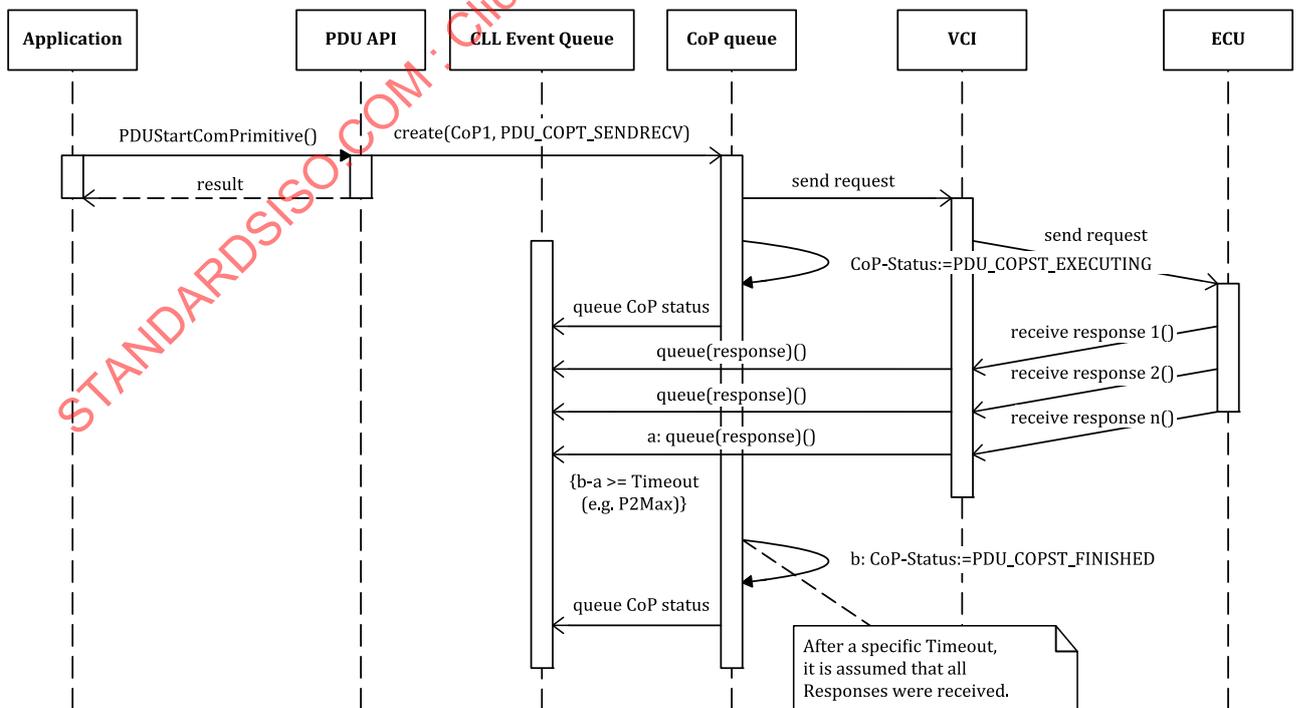
PDUGetEventItem actions may also be mixed with event callback function actions.

**8.2.6.4.4 Single request/multiple responses — Functional addressing**

Figures 11 to 14 show examples of single request/multiple response ComPrimitives for specific protocol implementations. The API calls are not shown in detail because they are already described in the figures above.



**Figure 11 — Example of single request, multiple responses — ISO 14230-3 functional addressing**



**Figure 12 — Example of single request, multiple responses — ISO 14229-1 finite periodic mode (e.g. time window)**

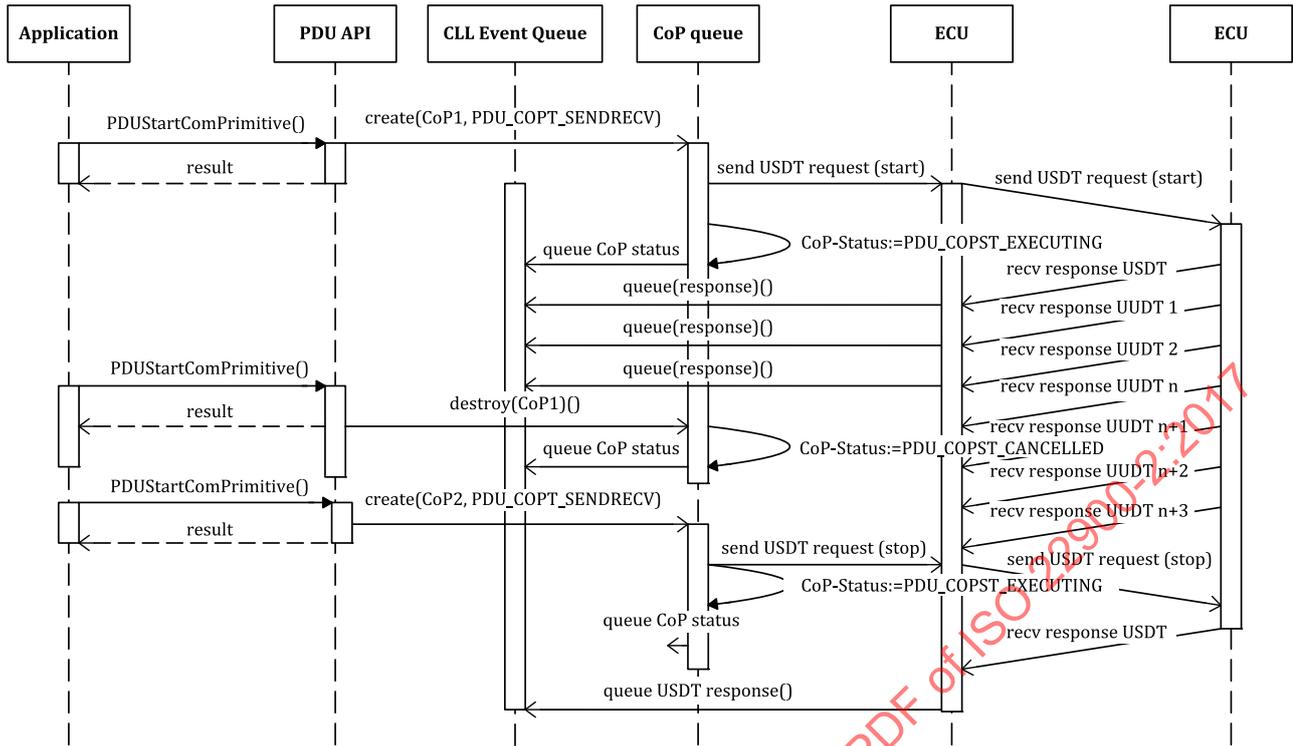


Figure 13 — Example of single request, multiple responses — ISO 14229-1 ReadDataByPeriodic-Identifier, type #2

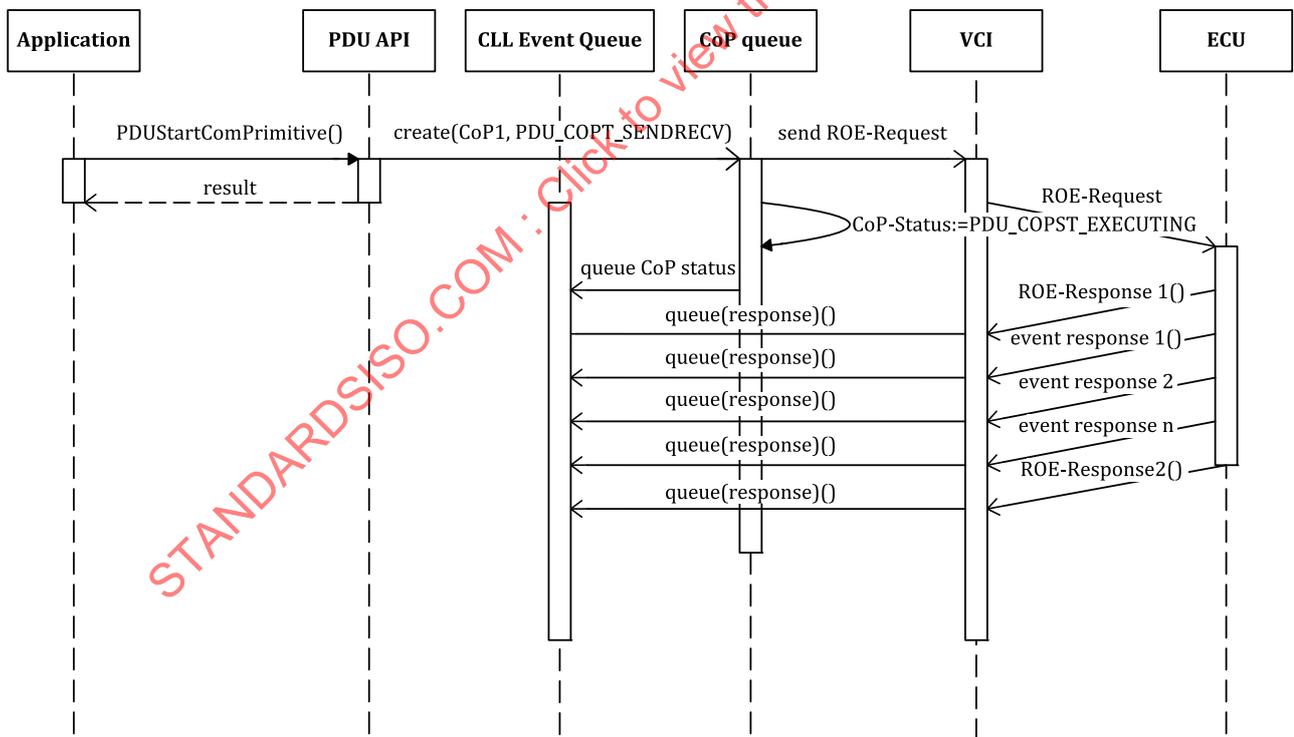


Figure 14 — Example of ISO 14229-1 ResponseOnEvent (RoE)

8.2.6.4.5 Single or multiple requests/no responses

This communication principle is used especially in on-board communication (e.g. CAN).

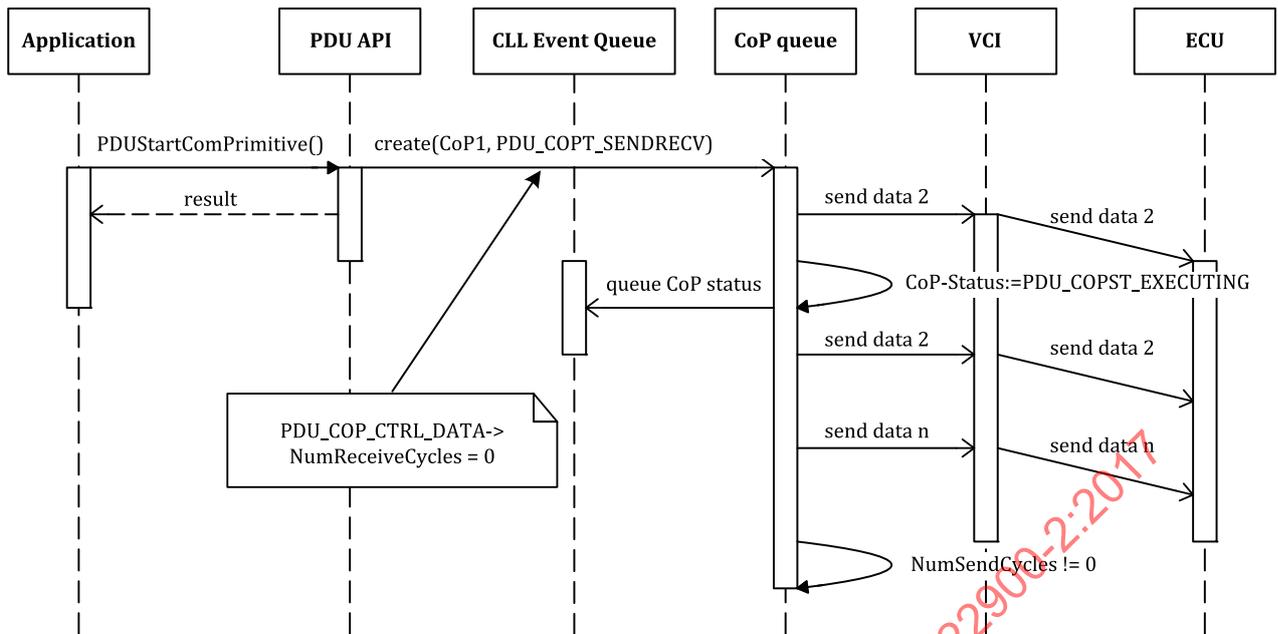


Figure 15 — Single or multiple requests, no responses

#### 8.2.6.4.6 No request/single or multiple responses

This communication principle is used especially in on-board communication (e.g. CAN). See Figure 16.

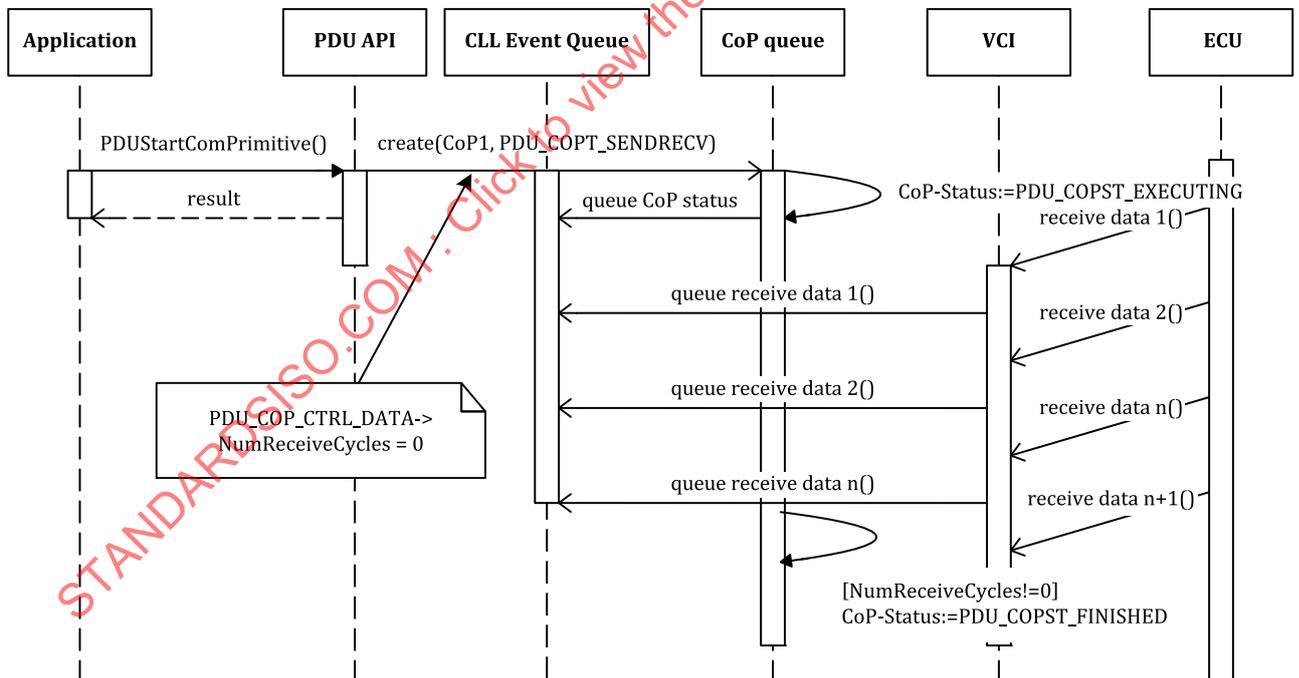


Figure 16 — No request, single or multiple responses

#### 8.2.6.5 Parallel execution of ComPrimitives

When the application starts a ComPrimitive with PDUStartComPrimitive, the ComPrimitive is put into the ComLogicalLink's internal ComPrimitive queue. Several ComPrimitives may be put into the queue sequentially. On the other end of the queue, the internal protocol driver fetches the ComPrimitives sequentially from the queue.

The protocol driver decides at which moment it is possible to start executing the next ComPrimitive. Depending on the type of protocol, it may be possible that one or more ComPrimitives can be executed in parallel by the protocol driver, according to certain rules, which are defined for the protocol.

**EXAMPLE** While a ComPrimitive delivering cyclic responses from the ECU is being executed, another single request/single response ComPrimitive can be executed.

It is advisable that the protocol driver be able to generally execute cyclic send/receive ComPrimitives in parallel. It depends on the implementation and how many of such ComPrimitives can be executed in parallel.

### 8.2.6.6 Cancelling a running ComPrimitive

It is possible to cancel a ComPrimitive using PDUCancelComPrimitive. If the ComPrimitive is still in the ComLogicalLink's internal ComPrimitive queue, it will just be removed from the queue. If it is already executing in the protocol driver, execution will be cancelled without any further interaction towards the ECU. The ComPrimitive status changes to PDU\_COPST\_CANCELLED. If the ComPrimitive has already reached a PDU\_COPST\_FINISHED state, no further action will be taken. A running ComPrimitive is never cancelled by starting a new ComPrimitive.

### 8.2.6.7 Destruction of ComPrimitives

A ComPrimitive completes when it has reached either the PDU\_COPST\_FINISHED state or the PDU\_COPST\_CANCELLED state, this status event item is placed on the ComLogicalLink event queue. Once a ComPrimitive has reached the FINISHED or CANCELLED state, no more result items will be queued for the ComPrimitive in the ComLogicalLink's event queue. The D-PDU API will destroy a ComPrimitive internally as soon as the last ComPrimitive status item is read from the event queue. After internal destruction of a ComPrimitive, no more operations can be executed related to this ComPrimitive (a PDU ERROR PDU\_ERR\_INVALID\_HANDLE will be returned for a function call referencing the destroyed ComPrimitive).

## 8.3 Tool integration

### 8.3.1 Requirement for generic configuration

The requirements to keep the D-PDU API close to existing standards on the one hand, and to take a generic and open approach on the other hand, make it difficult to limit the capabilities and therefore simplify the API. The configuration is not restricted to a predefined number of protocols, bus types and ComParams for SAE J2534-1. Instead, the D-PDU API shall support any protocols, bus types and ComParams as needed, independent of any definitions in a standard. Even the way equivalent protocols are implemented shall not be predefined by this document, i.e. there is no limitation in terms of which protocol and bus type ComParams to use (the only exception would be protocols and ComParams defined in SAE J2534-1).

Since the D-PDU API is kept open and generic, there is a major requirement to describe the MVCI protocol module's capabilities. Furthermore, once multiple MVCI protocol modules are bought and combined into one setup, the application running on the same setup requires a well-defined process to determine what is installed and what capabilities it has. There shall be a standardized entry point (similar to SAE J2534-1) and a navigation path through the configuration of the overall setup.

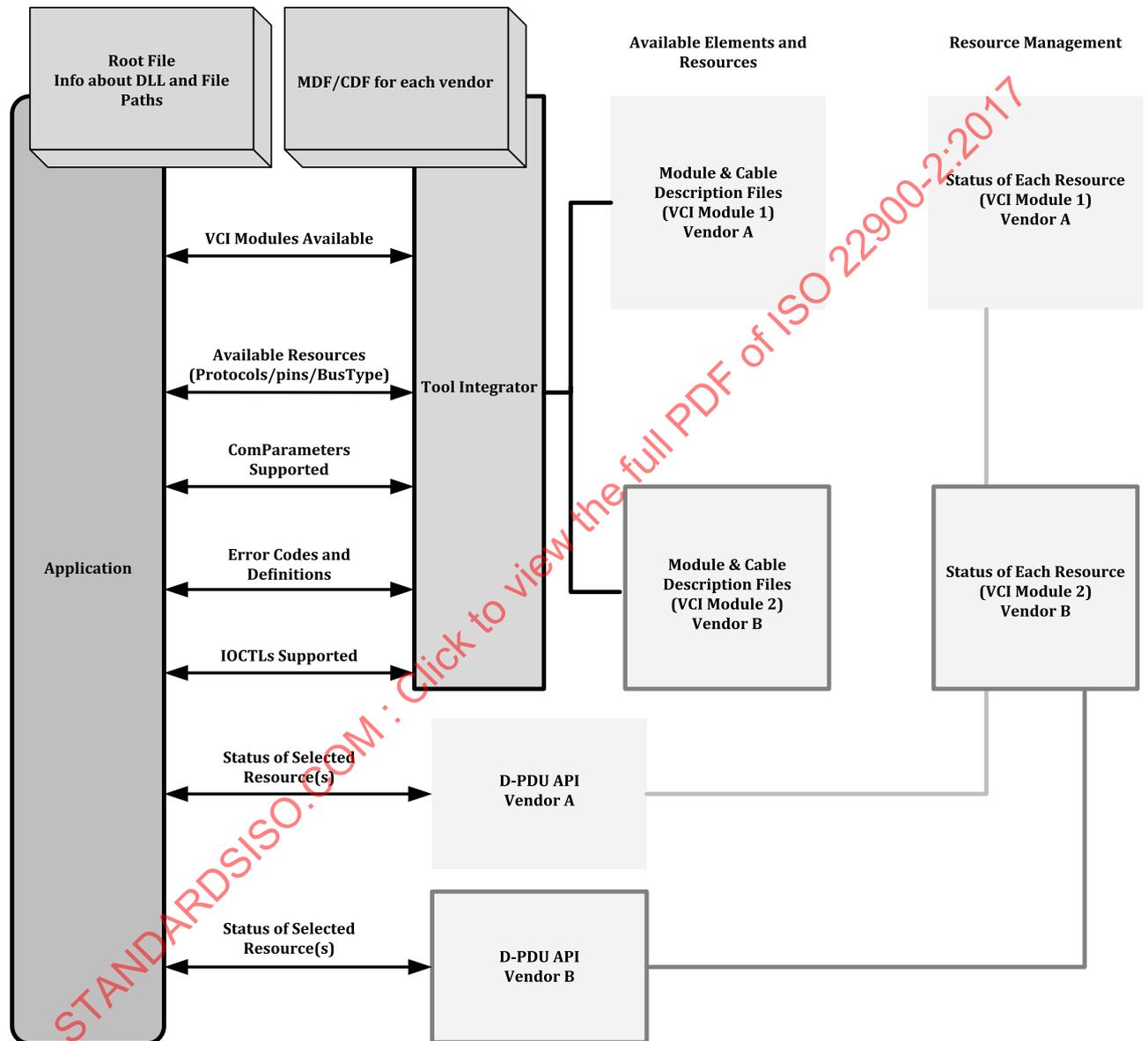
### 8.3.2 Tool integrator — Use case

An MVCI configuration used by an application may contain MVCI protocol modules and cables from different vendors. A client may purchase different MVCI protocol modules for their different implemented features (e.g. one MVCI protocol module has SAE J1850 support, the other MVCI protocol module has ISO 15765 support). Each MVCI protocol module vendor shall supply the following items: one or more module description files (MDF) (see F.2), one or more cable description files (CDF) (see F.3) and a D-PDU API DLL/Shared Library (see F.1). The application relies on the "tool integrator" to have knowledge of the overall system. Since there is no real plug-and-play functionality between different vendors, a knowledgeable Configuration Person would be required to use the Tool Integration Program to configure the overall system.

Figure 17 reflects a use case where two MVCI protocol modules are developed and introduced to the market by two independent vendors A and B at different times. A tool integrator, who is not necessarily one of the previous vendors, takes the MVCI protocol module A and its respective *MVCI protocol module Description File* from

vendor A. The tool integrator adds information describing where to find the D-PDU API implementation A and its description file. This information is stored in a file called *D-PDU API root description file*. In addition, the tool integrator knows what kind of cables shall be used, and therefore, adds information on all potential cables and how to map them onto the MVCI protocol module's resources. This information is stored in a file called *cable description file*.

At some point later in time, there is a need to extend this tool's capabilities, e.g. due to a merger or new vehicle protocols. The tool integrator adds the MVCI module description file B to his system and adapts the information describing where to find the D-PDU API implementation B and its description file. However, the integrator also has to adapt his cable information to reflect the correct mapping.



**Figure 17 — Use case reflecting a typical integration process for an MVCI setup**

Figure 18 shows an example where the mapping of MVCI protocol module pins to the Data Link Connector (DLC) pins are depicted. In addition, examples of the usage of these pins are shown, e.g. “K-Line 1” stands for an ISO 9141 connection. The tool integrator has to define whether a resource of an MVCI protocol module may be mapped onto a certain pin of his cable. For example:

- “K-Line 2” of MVCI protocol module A and “K-Line 1” of MVCI protocol module B may both be mapped onto pin “K-Line 1” of the DLC;
- “K-Line 1” of MVCI protocol module A may not be mapped onto pin “1” of the DLC;

— “SAE J1850” of MSCI protocol module B may be mapped onto pin “2” of the DLC.

As a result, the tool integrator is free to choose any kind of MSCI protocol module, but it is responsible for the overall system configuration. Since there is no real plug-and-play functionality available, the tool integrator has to take care of extending the configuration file deliverables.

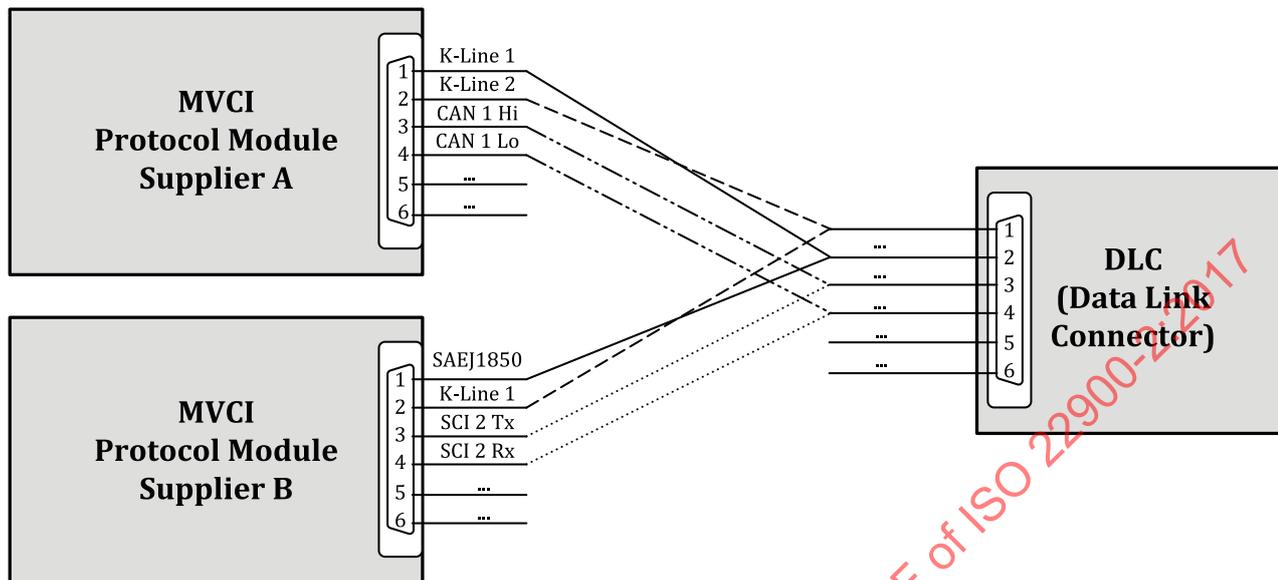


Figure 18 — Example of mapping MSCI protocol module pins to DLC pins

## 8.4 API functions — Interface description

### 8.4.1 Overview

This subclause describes all functions the D-PDU API offers to the application. For a description of the data types used, see 10.1.1. Definitions for the C/C++ qualifier are contained in 10.1.2. The function return values T\_PDU\_ERROR are listed in D.3. For each function, the return values are listed in the recommended order of precedence.

### 8.4.2 PDUConstruct

#### 8.4.2.1 Purpose

Initialize the D-PDU API library. The D-PDU API library determines the list of all available MSCI protocol modules and their supported resources. The D-PDU API library creates internal structures, including a resource table. The communication state is offline, i.e. no allocation of resources and no communication over vehicle interfaces take place.

#### 8.4.2.2 Behaviour

- a) Validates OptionStr.
- b) Detects all known and accessible MSCI protocol modules.
- c) Assigns Module IDs to all properly initialized MSCI protocol modules.
- d) Creates a PDU\_MODULE\_DATA list containing MSCI protocol modules, their module types, module handles, module status and additional vendor information strings. This function call will succeed even if no MSCI protocol modules are detected. In the case of no detection of any MSCI protocol modules, the call to PDUGetModuleIds will return a PDU\_MODULE\_ITEM with the number of entries set to zero (NumEntries = 0).

- e) All detected resources will be stored inside an internal resource table. This table will be the source for any resource query using the functions PDUGetModuleIds, PDUGetConflictingResources, PDUGetResourceIds, PDUGetResourceStatus, PDULockResource and PDUUnlockResource.

### 8.4.2.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUConstruct(CHAR8* OptionStr, void *pAPITag)
```

### 8.4.2.4 Parameters

**OptionStr** String containing a list of attributes and their values. An attribute and its corresponding value are to be separated by an >=< sign. The value needs to be put inside two >'< signs. Between pairs of attribute and value shall be at least one space character. Attributes and values are specific to a D-PDU API implementation.

When no option is to be set, the OptionStr can either be an empty string or NULL.

**pAPITag** An application defined tag value. Used in event callbacks which indicate status and errors and results for the D-PDU API library being used. This information can aid an application in determining which library is making the callback. For a more detailed description of using tags, see E.1.

### 8.4.2.5 Example

OptionStr = "UseCaching='TRUE' InterfaceCheck='FALSE' "

Table 10 specifies specific return values.

### 8.4.2.6 Return values

**Table 10 — PDUConstruct return values**

Definition	Description
PDU_ERR_SHARING_VIOLATION	Function called again without a previous destruct.
PDU_ERR_INVALID_PARAMETERS	At least one of the option attributes has invalid parameters.
PDU_ERR_VALUE_NOT_SUPPORTED	At least one of the option values is not being supported by the D-PDU driver.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.3 PDUDestruct

### 8.4.3.1 Purpose

Closes all open communication links and frees communication resources. Internal memory segments shall be freed and system-level drivers disconnected. Execution of PDUDestruct does not result in any communication on the vehicle interfaces. After execution of PDUDestruct, PDUConstruct may be called again.

### 8.4.3.2 Behaviour

- a) Checks internal resource table, determines any open communication links and closes them.

NOTE No communication (e.g. sending a "StopCommunication" request) takes place.

- b) De-initializes all MVCI protocol modules detected before with PDUConstruct.
- c) Closes all connections to connected MVCI protocol modules (see 8.4.30).
- d) Frees all internal memory resources.

**8.4.3.3 C/C++ prototype**

EXTERNC T\_PDU\_ERROR PDUdestruct()

**8.4.3.4 Parameters**

There are no parameters for this function.

Table 11 specifies specific return values.

**8.4.3.5 Return values**

**Table 11 — PDUdestruct return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.4.4 PDUIoctl**

**8.4.4.1 Purpose**

Generic approach to execute functions or set values related to an MVCI protocol module. An ID number identifies the function to be executed. The input and output values are defined as demanded by the function.

The I/O controls supported by a specific MVCI protocol module are defined within the MVCI module description file (see F.2).

8.5.1 provides a detailed view on the IOCTL API functions.

**8.4.4.2 Behaviour**

a) Validates all input parameters.

Required pointer parameters shall not be NULL. Non-required parameters shall not be checked for NULL.

b) Function takes an input data structure as allocated by the application.

c) Extracts required information from the input data structure and executes command.

d) Allocates and fills the output data in the call-by-reference variable pOutputData. The structure OutputData has to be freed by calling PDUDestroyItem from the application.

**8.4.4.3 C/C++ prototype**

EXTERNC T\_PDU\_ERROR PDUIoctl(UNUM32 hMod, UNUM32 hCLL, UNUM32 IoctlCommandId, PDU\_DATA\_ITEM \*pInputData, PDU\_DATA\_ITEM \*\*pOutputData)

**8.4.4.4 Parameters**

hMod Handle of the MVCI protocol module to be controlled by the specified I/O control command. hCLL shall be set to PDU\_HANDLE\_UNDEF to specify module related commands (see 8.5.1).

hCLL Handle of the ComLogicalLink to be controlled by the specified I/O control command.

IOCTLCommandId ID identifying the I/O control command. All IDs supported by a specific MVCI protocol module have to be defined within the MVCI module description file (see F.2).

pInputData	Input data item for specified command or NULL if no input data is required. The input data item shall be created and managed by the application. The structure of the data item is described in 10.1.4.3.
pOutputData	Call-by-reference place for storing the output data item pointer. If NULL, then no output data item will be allocated and filled by the D-PDU API implementation. If a valid address is provided, the D-PDU API implementation will allocate a PDU_DATA_ITEM item and fill in the output data of the specified command. A reference is stored in *pOutputData. After usage, the application shall free the allocated data item by calling PDU_DestroyItem. The structure of the data item is described in 10.1.4.3.

#### 8.4.4.5 Examples

PDUIoctl may be used to set programming voltages, reset the MVCI protocol module, or run a software update.

Table 12 specifies specific return values.

#### 8.4.4.6 Return values

**Table 12 — PDUIoctl return values**

Definition	Description
PDU_STATUS_NOERROR	Function call successful.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module or ComLogicalLink handle.
PDU_ERR_ID_NOT_SUPPORTED	ID of I/O control not supported by this MVCI protocol module.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) reference pointer for an IOCTL Command that expects one or both of the reference pointers to be valid (pInputData or pOutputData).
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDU_ModuleConnect function.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_MUX_RSC_NOT_SUPPORTED	The specified pin/resource are not supported by the MVCI protocol module.
PDU_ERR_VOLTAGE_NOT_SUPPORTED	The voltage is not supported by the MVCI protocol module.
PDU_ERR_RESOURCE_BUSY	Resource is busy; the application has to execute the command again.
PDU_ERR_RSC_LOCKED_BY_OTHER_CLL	The ComLogicalLink's resource is currently locked by another ComLogicalLink.
PDU_ERR_NO_CABLE_DETECTED	No cable is detected.
PDU_ERR_CABLE_UNKNOWN	Cable is unknown.
PDU_ERR_CLL_CONNECTED	CLL is already in the "online" state.
PDU_ERR_CLL_NOT_STARTED	Communications are not started on the ComLogicalLink yet. A Send ComPrimitive cannot be accepted in this state.
PDU_ERR_IP_PROTOCOL_NOT_SUPPORTED	IP protocol is not supported: e.g. IPv6 used as protocolVersion, but OS doesn't support IPv6 (or it is disabled).
PDU_ERR_ID_NOT_SUPPORTED	The MVCI device does not support a DoIP vehicle identification request
PDU_ERR_DOIP_AMBIGUOUS_LOGICAL_ADDRESS	Invalid or ambiguous logical address specified.
PDU_ERR_DOIP_RESPONSE_TIMEOUT	DoIPCtrlTimeout has occurred.

**8.4.5 PDUGetVersion**

**8.4.5.1 Purpose**

Function obtains version information from an MVCI protocol module.

**8.4.5.2 Behaviour**

a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

b) Fill out the PDU\_VERSION\_DATA structure allocated by the client application.

**8.4.5.3 C/C++ prototype**

EXTERNC T\_PDU\_ERROR PDUGetVersion( UNUM32 hMod, PDU\_VERSION\_DATA \*pVersionData )

**8.4.5.4 Parameters**

hMod Handle of the MVCI protocol module, for which the version information is to be requested.  
 pVersionData Call-by-reference place for storing the version information. The structure of the data item is described in 10.1.4.14.

Table 13 specifies specific return values.

**8.4.5.5 Return values**

**Table 13 — PDUGetVersion return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) pVersionData parameter.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.4.6 PDUGetStatus**

**8.4.6.1 Purpose**

Function obtains runtime information (status, life sign, etc.) from an MVCI protocol module, ComLogicalLink or ComPrimitive.

**8.4.6.2 Behaviour**

a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

b) Get the latest status information for the specified handle (Module or CLL or CoP) and store the information in the memory allocated by the client application.

- c) If the D-PDU API detects a PC software version out-of-date with the MVCI protocol module firmware, the status returned will be PDU\_MODST\_NOT\_READY.

#### 8.4.6.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetStatus( UNUM32 hMod, UNUM32 hCLL, UNUM32 hCoP, T_PDU_STATUS
*pStatusCode, UNUM32 *pTimestamp, UNUM32 *pExtraInfo)
```

#### 8.4.6.4 Parameters

hMod	Handle of MVCI protocol module for which the status code is to be requested. hCLL and hCoP shall be set to PDU_HANDLE_UNDEF to return status of only the module.
hCLL	Handle of ComLogicalLink for which the status code is to be requested. hCoP shall be set to PDU_HANDLE_UNDEF to return status of only the ComLogicalLink.
hCoP	Handle of ComPrimitive for which the status code is to be requested.
pStatusCode	Call-by-reference place for storing the status code (see D.1.4).
pTimestamp	Call-by-reference place for storing timestamp in microseconds.
pExtraInfo	Call-by-reference place for storing additional information. For ComPrimitives, pExtraInfo returns 0. For MVCI protocol modules and ComLogicalLinks, pExtraInfo contains additional information which is defined by the MVCI protocol module supplier. If no information is available, it shall return 0.

Table 14 specifies specific return values.

#### 8.4.6.5 Return values

**Table 14 — PDUGetStatus return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) pStatusCode, pTimestamp or pExtraInfo.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

#### 8.4.7 PDUGetLastError

##### 8.4.7.1 Purpose

Function obtains the code for the last error from an MVCI protocol module or ComLogicalLink. This function only returns the LAST error that occurred for the handle and has been added to this specification for SAE J2534-1 support. For proper error handling, a client application should be reading the error events stored in the respective event queues to retrieve a precise chronological order of events (see 8.4.19 for more information on retrieving event items).

By the time this function is called, the LAST error may already have been removed from the event queue. This also means that the provided hCoP handle may no longer be valid because the hCoP might already be finished and its associated resources freed.

##### 8.4.7.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Get the last error information for the specified handle (Module or CLL) and store the information in the memory allocated by the client application.
- c) In the case of an error being associated with a ComPrimitive, the ComPrimitive handle is returned with the error code.

**8.4.7.3 C/C++ prototype**

```
EXTERNC T_PDU_ERROR PDUGetLastError( UNUM32 hMod, UNUM32 hCLL, T_PDU_ERR_EVT *pErrorCode,
UNUM32 *phCoP, UNUM32 *pTimestamp, UNUM32 *pExtraErrorInfo)
```

**8.4.7.4 Parameters**

- hMod** Handle of MVCI protocol module for which the error code is to be requested. hCLL shall be set to PDU\_HANDLE\_UNDEF to return the last error of the MVCI protocol module.
- hCLL** Handle of ComLogicalLink for which the error code is to be requested.
- phCoP** If the last error pertained to a ComPrimitive, then phCoP will contain the handle of the ComPrimitive or else phCoP is set to PDU\_HANDLE\_UNDEF.
- pErrorCode** Call-by-reference place for storing the error code (see D.4). If no last error has been stored for the specified handle, then pErrorCode will contain PDU\_ERR\_EVT\_NOERROR.
- pTimestamp** Call-by-reference place for storing timestamp.
- pExtraErrorInfo** Call-by-reference place for storing extra error information. The ExtraErrorInfo code can be referenced from the MDF file.

Table 15 specifies specific return values.

**8.4.7.5 Return values**

**Table 15 — PDUGetLastError return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) pErrorCode, phCoP, pTimestamp or pExtraErrorInfo.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.4.8 PDUGetResourceStatus**

**8.4.8.1 Purpose**

Obtain resource status information from the D-PDU API. All resources whose status is to be retrieved are specified in the pResourceStatus structure. For each requested resource id, the corresponding resource status is reported back in the same structure.

The caller supplies a reference to a memory object that is an input/output resource status item (pResourceStatus). The caller-supplied memory object is of the type PDU\_RSC\_STATUS\_ITEM. The caller shall provide a pointer to the data item object, specifying the correct type (i.e. == PDU\_IT\_RSC\_STATUS) and the number of PDU\_RSC\_STATUS\_DATA objects for which status is to be retrieved. The D-PDU API shall validate the object and then fill in the output portions of the structure.

### 8.4.8.2 Behaviour

a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

b) Function takes pResourceStatus structure as allocated by the application.

c) Fills in the status information for each requested resource id.

### 8.4.8.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetResourceStatus(PDU_RSC_STATUS_ITEM *pResourceStatus)
```

### 8.4.8.4 Parameters

pResourceStatus Call-by-reference place for storing the status of all requested resource ids. The caller pre-fills the data structure prior to the call with the resource ids of interest. Data structure is described in 10.1.4.4.

Table 16 specifies specific return values.

### 8.4.8.5 Return values

**Table 16 — PDUGetResourceStatus return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) pResourceStatus, or one or more invalid resource ids.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.9 PDUCreateComLogicalLink

### 8.4.9.1 Purpose

#### 8.4.9.1.1 General

Function creates a ComLogicalLink for a given resource id. After creation, internal resources for this link are reserved and the communication state is offline, i.e. no vehicle communication is performed. However, the MVCI protocol module hardware shall be ready for communication at this point.

The most appropriate scheme for determining conflicting resources is to make use of MDF and CDF content that describes which resources conflict with one another. The D-PDU API gives additional support to the application by supplying a list of conflicts for a given resource (PDUGetConflictingResources) across multiple MVCI protocol modules of a single vendor.

The D-PDU API supports two schemes to create a ComLogicalLink. In the first scheme, the D-PDU API is provided with a set of resource items (bus type, protocol and pins), but no resource id. In the second scheme, the D-PDU API is provided with a known available resource for an MVCI protocol module.

#### 8.4.9.1.2 Unknown resource id scheme

For the unknown resource id scheme, the application is allowed to call PDUCreateComLogicalLink with a set of resources for that link (protocol id, bus type id, pin ids), without having previously checked whether that set of

resources is supported by this device, whether the resources are available or whether this request might conflict with another.

Although this scheme avoids the need for applications to check for availability and conflicts, it is expected that multi-connection applications can be supported. When an application requires multiple ComLogicalLinks, it may call PDUCreateComLogicalLink multiple times followed by corresponding calls to PDUConnect. The D-PDU API implementation has the opportunity to rearrange the resource requests implied by calls to PDUCreateComLogicalLink in order to avoid resource conflicts.

#### 8.4.9.1.3 Specific resource id scheme

For the specific resource id scheme, only the ResourceId parameter is used in PDUCreateComLogicalLink. This id would be obtained by either reading the MDF file or calling PDUGetResourceIds, and could optionally be checked for availability and conflicts, at the discretion of the application (PDUGetResourceStatus and PDUGetConflictingResources).

#### 8.4.9.2 Behaviour

- a) Validate all input parameters.

Required pointer parameters shall not be NULL. Non-required parameters shall not be checked for NULL.

- b) Check if resource is still available.
- c) If available, mark resource as “in use” in the resource table.
- d) Only the ComLogicalLink which uses PDUlockResource will get exclusive rights to modify the physical ComParams for the resource (see Table 21).
- e) Therefore, the default behaviour for ComLogicalLinks which share a physical resource is that they may all modify the physical ComParams. Since there is only one set of physical ComParams, each ComLogicalLink sharing a physical resource will read the last values set.
- f) The event status item (PDU\_CLLST\_OFFLINE) will NOT be generated on creation of a ComLogicalLink. It shall be assumed by the application that OFFLINE is the initial status after creation. This is required since the event callback function has not yet been defined by the application for the ComLogicalLink.

##### 8.4.9.2.1 Behaviour — Use cases

When a ComLogicalLink changes status, a status event item is generated (see D.1.4). The following list describes each status change use case.

- a) Use case: CLL State = PDU\_CLLST\_OFFLINE

This is the initial state of the ComLogicalLink on creation (PDUCreateComLogicalLink). NO status event item is generated on the initial creation of a ComLogicalLink because the callback registration requires the ComLogicalLink handle (hClL) (see 8.4.21). The ComLogicalLink shall be in the state PDU\_CLLST\_ONLINE to allow any ComPrimitive queuing (see 8.4.11 and 8.4.17).

- b) Use case: CLL State Change = (any state -> PDU\_CLLST\_OFFLINE)

The ComLogicalLink transitions to PDU\_CLLST\_OFFLINE from any other ComLogicalLink state on a successful function call to PDUDisconnect or on a loss of communication to a module. All ComPrimitives currently executing (i.e. periodic) and all ComPrimitives in the CoP queue will be cancelled (PDU\_COPST\_CANCELLED). A status event item, PDU\_COPST\_CANCELLED, is generated for each active CoP for the ComLogicalLink. The orders of events under the case of losing communications to a module are: PDU\_ERR\_EVT\_LOST\_COMM\_TO\_VCI, PDU\_COPST\_CANCELLED, PDU\_CLLST\_OFFLINE and PDU\_MODST\_NOT\_AVAIL.

- c) Use case: CLL State Change = (PDU\_CLLST\_OFFLINE -> PDU\_CLLST\_ONLINE)

The ComLogicalLink changes state from PDU\_CLLST\_OFFLINE to PDU\_CLLST\_ONLINE after a successful call to PDUConnect.

- d) Use case: CLL State Change = (PDU\_CLLST\_ONLINE -> PDU\_CLLST\_COM\_STARTED)

The ComLogicalLink changes state from PDU\_CLLST\_ONLINE to PDU\_CLLST\_COM\_STARTED after successful execution of the ComPrimitive of type PDU\_COPT\_STARTCOMM (see PDUStartComPrimitive). If tester present handling is enabled (see CP\_TesterPresentHandling), the message is sent immediately, prior to the initial tester present cyclic time (CP\_TesterPresentTime). After initial transmission the periodic tester present cycles begin.

NOTE Tester Present messages are only enabled in the state PDU\_CLLST\_COM\_STARTED.

- e) Use case: CLL State Change = (PDU\_CLLST\_COM\_STARTED -> PDU\_CLLST\_ONLINE)

The ComLogicalLink changes state from PDU\_CLLST\_COM\_STARTED to PDU\_CLLST\_ONLINE after successful execution of the ComPrimitive of type PDU\_COPT\_STOPCOMM.

### 8.4.9.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUCreateComLogicalLink(UNUM32 hMod, PDU_RSC_DATA *pRscData, UNUM32
resourceId, void *pCllTag, UNUM32 *phCLL, PDU_FLAG_DATA *pCllCreateFlag )
```

### 8.4.9.4 Parameters

hMod	Handle of MvCI protocol module.
pRscData	Resource Data Objects used to define the settings for a ComLogicalLink. Data structure described in 10.1.4.8.  Unknown Resource Scheme: The pRscData shall not be NULL. All elements in pRscData are checked for validity. The resourceId parameter shall be set to PDU_ID_UNDEF.  Specific Resource Id Scheme: pRscData shall be set to NULL. The resourceId parameter is used instead and shall be valid.
resourceId	Resource Id which maps to the resource data objects defined in the D-PDU API Resource Table. This information is used to define the settings for the ComLogicalLink.  Unknown Resource Scheme: The resourceId parameter shall be set to PDU_ID_UNDEF. The pRscData shall not be NULL.  Specific Resource Id Scheme: The resourceId parameter shall be valid. The Resource Objects are selected from the D-PDU API Resource Table using the "resourceId" parameter. pRscData shall be set to NULL.
pCllTag	An application defined tag value. Used in event callbacks which indicate status and errors and results for the ComLogicalLink. For a more detailed description of using tags, see E.1.
phCLL	Call-by-reference place for storage of ComLogicalLink handle.
pCllCreateFlag	Call-by-reference place for storage of flag bits used in creating a ComLogicalLink. See 10.1.4.13 and Table D.6.

Table 17 specifies specific return values.

## 8.4.9.5 Return values

Table 17 — PDUCreateComLogicalLink return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	Invalid NULL pointer for phCLL, invalid NULL pointer for pCllCreateFlag or invalid resource id.
PDU_ERR_RESOURCE_BUSY	Resource is busy; the application has to execute the command again.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.10 PDUDestroyComLogicalLink

## 8.4.10.1 Purpose

Destroys the given ComLogicalLink.

## 8.4.10.2 Behaviour

- a) Validate all input parameters.
- b) This function destroys all unread items in the ComLogicalLink's event queue.
- c) All ComPrimitives for the ComLogicalLink are cancelled without generating a PDU\_COPST\_CANCELLED event (because the handle of the ComLogicalLink is no longer valid).
- d) Any items that have been previously "read" by using PDUGetEventItem are still "reserved" by the application. A call to PDUDestroyItem is still necessary to release any "reserved" memory.
- e) The ComLogicalLink is disconnected from the physical resource.
- f) The D-PDU API releases the physical resource from the ComLogicalLink.
  - 1) Shared resource behaviour:
    - If the resource is shared by another ComLogicalLink, then the physical ComParams remain unchanged.
    - The physical resource remains connected to the physical bus if the sharing ComLogicalLinks are NOT in the PDU\_CLLST\_OFFLINE state.
    - If the ComLogicalLink had a lock on the physical ComParams and/or the physical transmit queue, the lock will automatically be removed. When the change of lock status occurs, an information callback is made to the shared ComLogicalLinks indicating a change in lock status (see Table 21, Table 22 and PDU\_IT\_INFO).
  - 2) Not shared resource behaviour:
    - If the resource is not shared by other ComLogicalLinks, then it becomes available to the whole system (PDU\_RSCST\_AVAIL\_NOT\_IN\_USE).

g) The hCLL handle is no longer valid. No event items are queued during this function call.

### 8.4.10.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUDestroyComLogicalLink(UNUM32 hMod, UNUM32 hCLL)
```

### 8.4.10.4 Parameters

hMod Handle of MSCI protocol module.

hCLL Handle of ComLogicalLink to be destroyed.

Table 18 specifies specific return values.

### 8.4.10.5 Return values

**Table 18 — PDUDestroyComLogicalLink return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MSCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MSCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MSCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.11 PDUConnect

### 8.4.11.1 Purpose

Physically connect a ComLogicalLink to the vehicle interface. The ComLogicalLink shall be in the communication state “offline” when the function is called. After execution, the communication state changes from “offline” to “online”. This function call is required before any communication to the vehicle ECU can take place.

### 8.4.11.2 Behaviour

- a) Validate all input parameters.
- b) Put working buffer ComParams into the active buffer. Locked physical ComParams are not changed. When the condition occurs that any physical ComParam is different than the locked physical ComParam, an error event item is generated for the CLL (PDU\_ERR\_EVT\_RSC\_LOCKED) indicating that one or more physical ComParams do not match the actual list of physical ComParams. The ComLogicalLink will still transition to the ONLINE state even if the physical ComParams do not match.
- c) Put the working table of Unique Response Identifiers into the active table.
- d) Configure and enable the ComLogicalLink filters. Use the URID table for the filter configuration unless the client application has configured filters prior to PDUConnect using any of the PDU\_IOCTL function calls: PDU\_START\_MSG\_FILTER, PDU\_CLEAR\_MSG\_FILTER and PDU\_STOP\_MSG\_FILTER. The configuration by the client application overrides any D-PDU API internal configuration using the URID table.
- e) Physically connect to the vehicle bus.

**NOTE** A physical connection to the vehicle bus is not done at the time of PDUConnect when a PDU\_COPT\_STARTCOMM is required to determine proper physical bus parameters, e.g. connection for ISO\_OBD\_on\_ISO\_15765\_4 (OBD on CAN). (See J.1.3.4.) Improper configuration of physical ComParams could cause significant bus errors.

f) Set the state of the CLL to PDU\_CLLST\_ONLINE and generate an event indicating the new state.

### 8.4.11.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUConnect(UNUM32 hMod, UNUM32 hCLL)
```

### 8.4.11.4 Parameters

hMod Handle of MVCI protocol module.

hCLL Handle of ComLogicalLink to be connected to the vehicle interface.

Table 19 specifies specific return values.

### 8.4.11.5 Return values

Table 19 — PDUConnect return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_CLL_CONNECTED	CLL is already in the “online” state.
PDU_ERR_DOIP_AMBIGUOUS_LOGICAL_ADDRESS	Denied to connect a DoIP LogicalLink with a logical address which is identical for multiple DoIP entities inside a DoIP MVCI module representing a collection of DoIP entities.
PDU_ERR_DOIP_ROUTING_ACTIVATION_FAILED	DoIP Routing activation failed. Used if no other more specific error code related to routing activation is available.
PDU_ERR_DOIP_ROUTING_ACTIVATION_AUTHENTICATION_FAILED	DoIP Routing activation denied due to response code 0x04 indicating missing authentication.
PDU_ERR_DOIP_ROUTING_ACTIVATION_INVALID_SOURCE_ADDRESS	Routing activation denied due to response code 0x00 indicating an unknown or invalid source address.
PDU_ERR_DOIP_ROUTING_ACTIVATION_NO_DATA_SOCKET_AVAILABLE	Routing activation denied due to response code 0x01 indicating too many registered and active TCP clients.
PDU_ERR_DOIP_ROUTING_ACTIVATION_SOURCE_ADDRESS_CHANGED	Routing activation denied due to response code 0x02 indicating that a different source address has already been activated before on this TCP connection.
PDU_ERR_DOIP_ROUTING_ACTIVATION_SOURCE_ADDRESS_IN_USE	Routing activation denied due to response code 0x03 indicating that the same source address is already registered and active on a different TCP connection.
PDU_ERR_DOIP_ROUTING_ACTIVATION_CONFIRMATION_REJECTED	Routing activation denied due to response code 0x05 indicating a rejected confirmation.
PDU_ERR_DOIP_ROUTING_ACTIVATION_TYPE_UNSUPPORTED	Routing activation denied due to response code 0x06 indicating an unsupported routing activation type.
PDU_ERR_DOIP_ROUTING_ACTIVATION_RESPONSE_CODE_UNKNOWN	Routing activation denied due to an unknown (i.e. manufacturer specific) routing activation response code.
PDU_ERR_DOIP_ROUTING_ACTIVATION_RESPONSE_TIMEOUT	Time out while waiting for routing activation response.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.12 PDUDisconnect

### 8.4.12.1 Purpose

Physically disconnect ComLogicalLink from vehicle interface. The ComLogicalLink shall be in the communication state “online” or “com\_started” when the function is called. After execution, the communication state changes to “offline”. After calling the function, no more communication to the vehicle ECU may take place.

### 8.4.12.2 Behaviour

- a) Validate input parameters.
- b) Prevent initiation of new ComPrimitives by marking ComLogicalLink as “disconnected” in the internal resource table.
- c) Cancel all active (executing) ComPrimitives. Cancel all idle ComPrimitives from the ComPrimitive queue.
- d) All ComParam values and ComLogicalLink filters are preserved for a future reconnection (i.e. they are not returned to their default values).
- e) Physically disconnect the ComLogicalLink from the resource. The resource is still reserved by the ComLogicalLink until a PDUDestroyComLogicalLink function call has been completed.

Shared resource behaviour:

- if the resource is shared by another ComLogicalLink, then the physical ComParams remain unchanged;
- if the resource is shared by another ComLogicalLink, then the physical resource remains connected to the physical bus if the sharing ComLogicalLinks are NOT in the PDU\_CLLST\_OFFLINE state;
- if the ComLogicalLink had a lock on the physical ComParams and/or the physical transmit queue, the lock will automatically be removed. When the change of lock status occurs, an information callback is made to the shared ComLogicalLinks indicating a change in lock status (see Table 21, Table 22 and PDU\_IT\_INFO).

### 8.4.12.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUDisconnect(UNUM32 hMod, UNUM32 hCLL)
```

### 8.4.12.4 Parameters

hMod    Handle of MVCI protocol module.  
hCLL    Handle of ComLogicalLink to be disconnected from vehicle interface.  
Table 20 specifies specific return values.

### 8.4.12.5 Return values

**Table 20 — PDUDisconnect return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_CLL_NOT_CONNECTED	ComLogicalLink is not connected.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.13 PDULockResource

#### 8.4.13.1 Purpose

PDULockResource will allow exclusive privileges for a ComLogicalLink on a physical resource. This function can be used by an application which wants to have physical bus protection from multiple ComLogicalLinks which share the physical resource.

If the function call is successful, the ComLogicalLink has been granted exclusive rights to the physical resource based on the lockMask value (see Table D.2).

Monitoring or receiving messages from a physical resource is not influenced nor affected by locking.

The call to PDULockResource will fail if another ComLogicalLink already has acquired the requested lock type on the same resource.

NOTE PDULockResource has no effect for ComLogicalLinks using the ISO\_14229\_5\_on\_ISO\_13400\_2 protocol.

#### 8.4.13.2 Behaviour

- a) Validate all input parameters.
- b) Check for other locks and currently active transmissions on the resource. Resume with c) if this is not the case. Otherwise, return with value PDU\_ERR\_RSC\_LOCKED or PDU\_ERR\_FCT\_FAILED.
- c) Set the status of the ComLogicalLink's resource.

#### 8.4.13.3 Behaviour — Use cases

##### a) Transmit Queue Lock

A lock on the transmit queue will force a SUSPEND\_TX\_QUEUE to all other ComLogicalLinks sharing the physical resource. Any new ComLogicalLink are assumed to have their ComPrimitive queue in the SUSPEND\_TX\_QUEUE mode. See PDUIOCTL function PDU\_IOCTL\_SUSPEND\_TX\_QUEUE. When the lock on the transmit queue is released, a RESUME\_TX\_QUEUE is sent to all ComLogicalLinks sharing the physical resource (see 8.5.5).

NOTE When a ComPrimitive queue is suspended, tester present messages will also be stopped if enabled (see CP\_TesterPresentHandling).

##### b) Physical ComParam Lock

A lock on the physical ComParams (ComParam ActiveBuffer) will not terminate any ongoing transmissions. However, calls to PDU\_COPT\_UPDATEPARAM for physical ComParams on a second ComLogicalLink will cause an error event for the second CLL (PDU\_ERR\_EVT\_RSC\_LOCKED).

##### c) Automatic unlocking

An automatic unlock occurs during a PDUDestroyComLogicalLink and PDUDisconnect function calls.

##### d) Change of lock status

When the lock status of a resource changes, an information callback (PDU\_IT\_INFO) is made to other ComLogicalLinks that are sharing the physical resource, informing them that a lock status has changed. A client application can then call PDUGetResourceStatus to determine the current lock state of the physical resource (see Table D.1).

#### 8.4.13.4 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDULockResource(UNUM32 hMod, UNUM32 hCLL, UNUM32 LockMask)
```

### 8.4.13.5 Parameters

hMod Handle of MVCI protocol module.

hCLL Handle of ComLogicalLink to be granted exclusive access to its resource.

LockMask Bit encoded mask to request a type of exclusive privilege to a physical resource (see Table D.2).

Table 21 specifies specific return values.

### 8.4.13.6 Return values

**Table 21 — PDULockResource return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_RSC_LOCKED	The requested resource is already in the "locked" state.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.14 PDUUnlockResource

#### 8.4.14.1 Purpose

PDUUnlockResource releases the lock type on the resource the ComLogicalLink is connected to as long as the lock type has previously been locked by the same ComLogicalLink.

If the function call is successful, the lock will be released for the resource.

#### 8.4.14.2 Behaviour

- a) Validate all input parameters.
- b) Set the status of the ComLogicalLink's resource.
- c) When the lock status of a resource changes, an information callback (PDU\_IT\_INFO) is made to other ComLogicalLinks that are sharing the physical resource, informing them that a lock status has changed. A client application can then call PDUGetResourceStatus to determine the current lock state of the physical resource (see Table D.1).

#### 8.4.14.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUUnlockResource(UNUM32 hMod, UNUM32 hCLL, UNUM32 LockMask)
```

#### 8.4.14.4 Parameters

hMod Handle of MVCI protocol module.

hCLL Handle of ComLogicalLink unlocking the resource.

LockMask Bit encoded mask to release the type of exclusive privilege to a physical resource (see Table D.2).

Table 22 specifies specific return values.

## 8.4.14.5 Return values

Table 22— PDUUnlockResource return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_RSC_LOCKED_BY_OTHER_CLL	The requested resource is locked by a different ComLogicalLink.
PDU_ERR_RSC_NOT_LOCKED	The resource is already in the unlocked state.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.15 PDUGetComParam

## 8.4.15.1 Purpose

PDUGetComParam obtains a communication or bus ComParam out of the working buffer. The values in the working buffer correspond to the values that would be set in the MVCI protocol module when it has executed all previous Communication Primitives that currently exist in the ComPrimitive queue and any changes previously made by this logical link via PDUSetComParam. The sequence diagrams in Figures 19 to 20 and the buffer diagram in Figure 22 illustrate the relationship between the active buffer, working buffer and communication primitives.

## 8.4.15.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Allocate memory for the PDU\_PARAM\_ITEM result.  
 c) Fill out the ComParam information from the ComParam working buffer.

## 8.4.15.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetComParam(UNUM32 hMod, UNUM32 hCLL, UNUM32 ParamId,
PDU_PARAM_ITEM **pParamItem)
```

## 8.4.15.4 Parameters

- hMod Handle of MVCI protocol module.  
 hCLL Handle of ComLogicalLink for which the ComParam is to be requested.  
 ParamId ID value of the ComParam, which is to be requested. The MVCI protocol module supplier provides the ComParam ID values in the MDF.  
 pParamItem Call-by-reference place for storing the item with the requested ComParam from the MDF, according to the specified ParamId. The item is allocated by the D-PDU API and has to be released after use from the application by calling the function PDUDestroyItem(). Data structure described in 10.1.4.5.

Table 23 specifies specific return values.

## 8.4.15.5 Return values

Table 23 — PDUGetComParam return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_COMPARAM_NOT_SUPPORTED	ComParam is not supported, e.g. because it is of type PDU_PS_ECU, PDU_PS_OPTIONAL or PDU_PC_UNIQUE_ID.
PDU_ERR_INVALID_PARAMETERS	Invalid ComParam ID or Invalid NULL pointer for pParamItem.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.15.6 Example for ComParam Buffer Operation

Table 24 describes example operations of ComParam buffer control. Diagram representations of the sequence steps are reflected in Figures 19 to 23.

Table 24 — Example for ComParam buffer operation

Sequence (figure)	Action	Working buffer (P2)	Active buffer (P2)	Notes
1 (1/5)	PDUCreateCommLogicalLink	Defaults	Defaults	This will be the state of the ComParams. (P2 will equal the default value in both buffers.)
2 (1/5)	PDUGetComParam	Defaults	Defaults	Obtains ComParam out of the working buffer.
3 (1/5)	PDUSetComParam (P2=50 ms)	50 ms	Defaults	Modifies the ComParam in the working buffer.
4 (1/5)	PDUConnect	50 ms →	50 ms	This function call copies the ComParams in the working buffer to the active buffer.
5 (1/5)	PDUStartComPrimitive1 (PDU_COPT_STARTCOMM)	50 ms	50 ms	This Com Primitive is now placed on the queue with an association to the active buffer settings (P2 = 50 ms).
6 (2/5)	PDUSetComParam (P2=30 ms)	30 ms	50 ms	Modifies the ComParam in the working buffer.
7 (2/5)	PDUStartComPrimitive2 (PDU_COPT_SENDRECV) with TempParamUpdate flag set	30 ms	50 ms	This Com Primitive is now placed on the queue with an association to the working buffer settings (P2 = 30 ms).
7.1 (2/5)	Return of function call PDUStartComPrimitive2	50 ms	← 50 ms	This specification requires that the working buffer is restored to active buffer upon the return of the PDUStartComPrimitive function call with the TempParamUpdateFlag set to 1. (working buffer P2 is reset to 50 ms.)
8 (2/5)	PDUGetComParam	50 ms	50 ms	Gets the ComParam from the working buffer (P2 = 50 ms).
9 (3/5)	PDUSetComParam (P2=20 ms)	20 ms	50 ms	Modifies the ComParam in the working buffer (P2 = 20 ms).
10 (3/5)	PDUGetComParam	20 ms	50 ms	Gets the ComParam from the working buffer (P2 = 20 ms).

Table 24 — (continued)

Sequence (figure)	Action	Working buffer (P2)	Active buffer (P2)	Notes
11 (3/5)	PDUStartCom Primitive3 (COPT_SENDRECV)	20 ms	50 ms	This Com Primitive is now placed on the queue with an association to the active buffer settings (P2 = 50 ms).
12 (3/5)	PDUStartCom Primitive4 (COPT_UPDATEPARAM)	20 ms →	20 ms	This function call copies the working buffer into the active buffer after which it is placed on the queue. (Active buffer P2 is set to 20 ms.)
13 (3/5)	PDUSetComParam (P2=40 ms)	40 ms	20 ms	Modifies the ComParam in the working buffer (P2 = 40 ms).
14 (3/5)	PDUGetComParam	40 ms	20 ms	Gets the ComParam from the working buffer (P2 = 40 ms).
15 (3/5)	PDUStartCom Primitive5 (COPT_SENDRECV)	20 ms	20 ms	This Com Primitive is now placed on the queue with an association to the active buffer settings (P2 = 20 ms).
16 (5/5)	PDUStartCom Primitive6 (COPT_SENDRECV) with TempParamUpdate flag set	40 ms	20 ms	This Com Primitive is now placed on the queue with an association to the working buffer settings (P2 = 40 ms).

#### 8.4.15.7 Example for ComParam buffer operation for PDUConnect

Figure 19 shows the sequence between a PDUCreateComLogical function call and a PDUConnect function call. All PDUSetComParam calls store the ComParam information in the working buffer. The working buffer is copied to the active buffer when a PDUConnect function is called. In the example, the P2 ComParam was set to its initial default value when the ComLogicalLink was created. The value was changed by a PDUSetComParam function and moved to the active buffer on a PDUConnect. The first ComPrimitive (CoP1) gets the active buffer of ComParams attached to it (i.e. the P2 ComParam value is 50 ms).

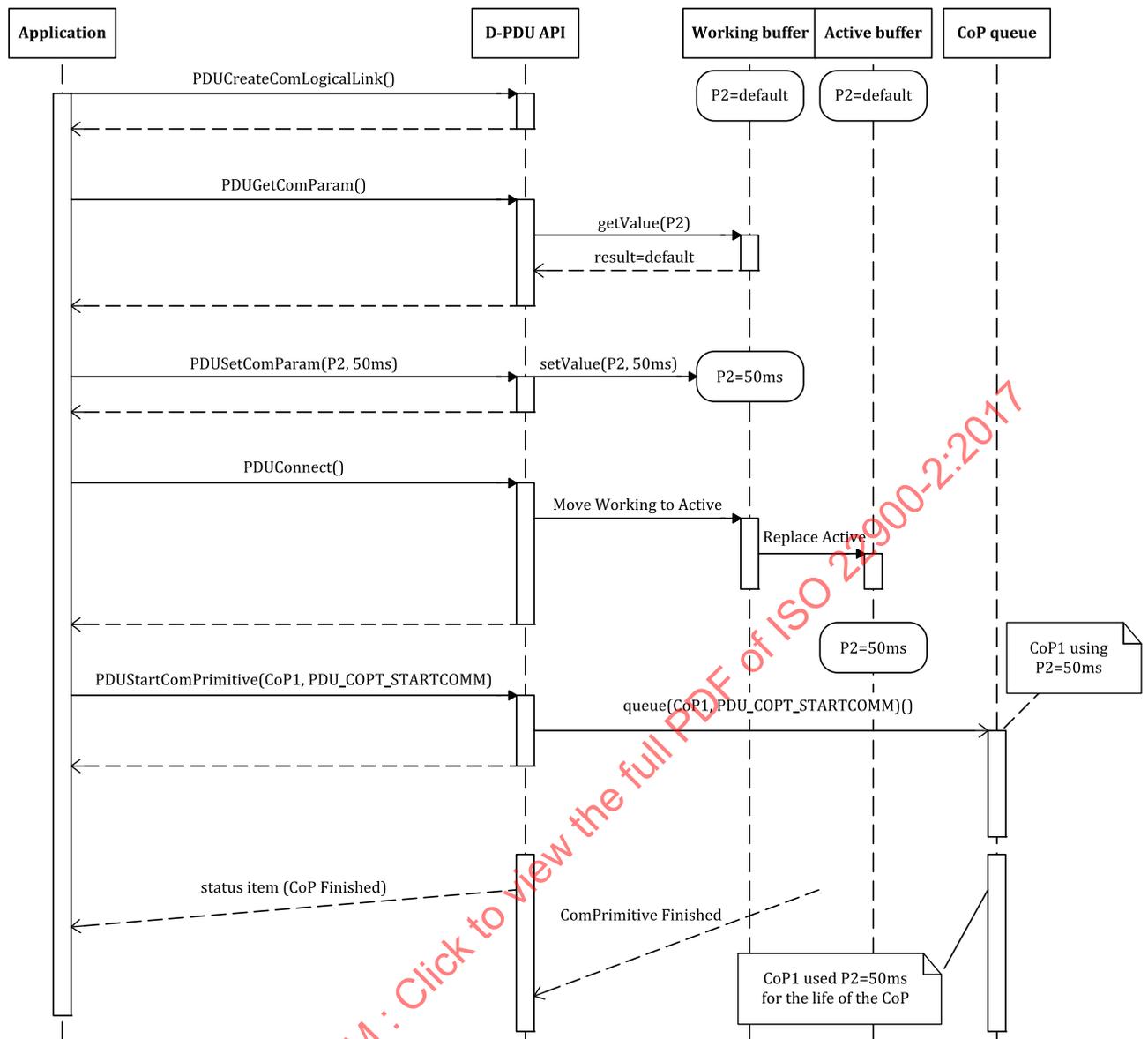


Figure 19 — Example for ComParam buffer operation on a PDUconnect (1 of 5)

#### 8.4.15.8 Example for ComParam buffer operation using TempParamUpdate flag

Figure 20 shows the sequence of operations when the TempParamUpdate flag is set to 1 in PDU\_COP\_CTRL\_DATA for a ComPrimitive. In the example, the P2 ComParam is set to 50 ms in the active buffer. The working buffer contains a P2 ComParam value of 30 ms. When the ComPrimitive (CoP2) is started with the TempParamUpdate flag set to 1, the ComPrimitive gets the working buffer of ComParams attached to it. The ComPrimitive (CoP2) uses the P2 ComParam value of 30 ms until it is finished (i.e. status is set to PDU\_COPST\_FINISHED). But the working buffer gets restored to the active buffer settings immediately after the ComPrimitive was placed on the CoP Queue (i.e. any changes made to the working buffer are cleared immediately after the PDUStartComPrimitive function is called with the TempParamUpdate flag set to 1).

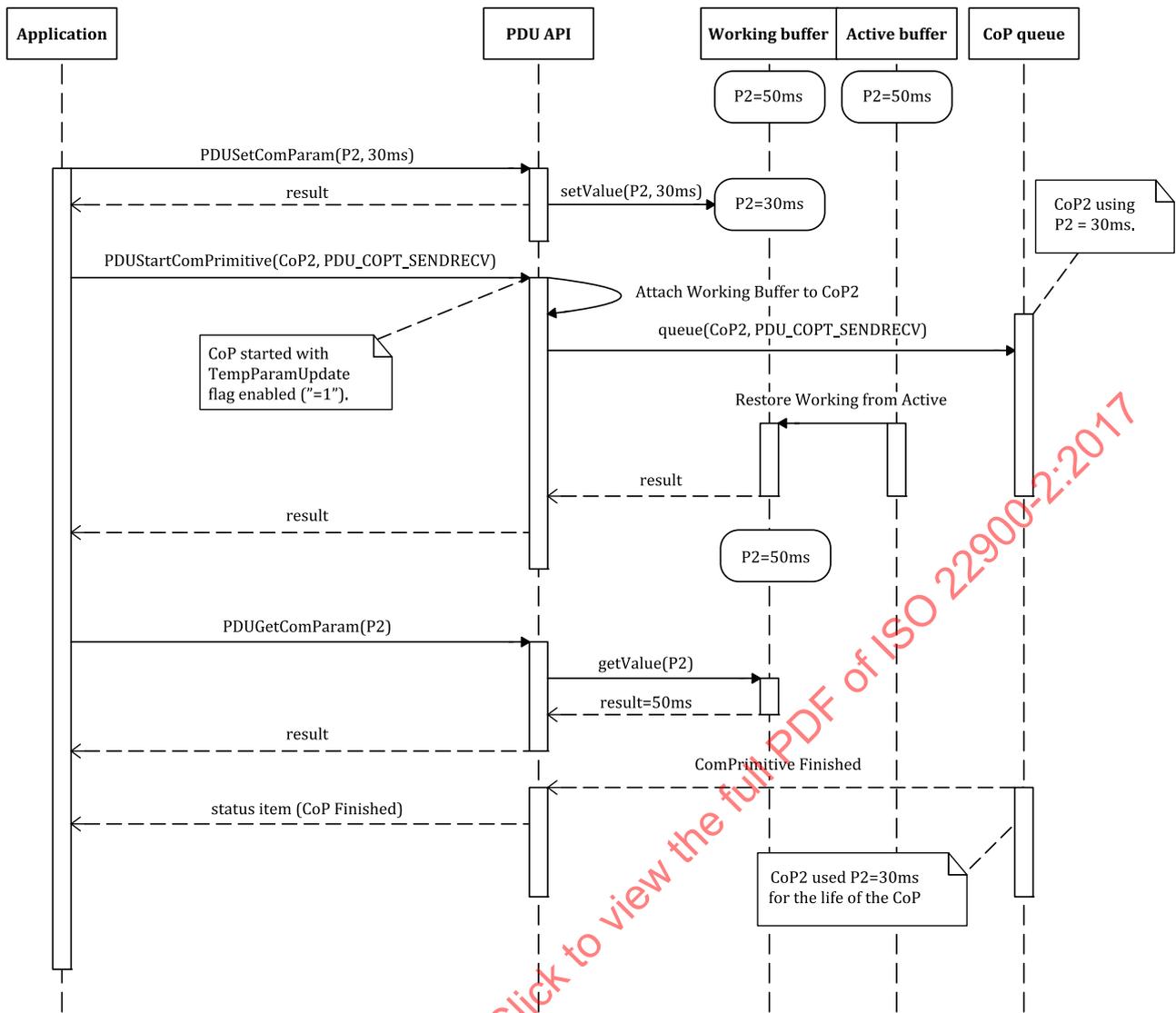


Figure 20 — Example for ComParam buffer operation using TempParamUpdate flag (2 of 5)

8.4.15.9 Example for ComParam buffer operation using PDU\_COPT\_UPDATEPARAM

Figure 21 shows the sequence of operations to permanently move a ComParam from the working buffer to the active buffer and how this move does NOT affect other ComPrimitives which are currently executing or already in the CoP queue. In the example, ComPrimitive (CoP3) is using a set of ComParams with P2 set to 50 ms. The application wants to change this value for all future ComPrimitives to 20 ms. The working buffer gets set with P2 ComParam value of 20 ms. When the ComPrimitive (CoP4) is started with the CopType of PDU\_COPT\_UPDATEPARAM, the working buffer is moved to the active buffer. This change does not affect the CoP3 ComPrimitive. The CoP4 ComPrimitive finishes immediately (i.e. status is set to PDU\_COPST\_FINISHED). Therefore, when the next ComPrimitive is started (CoP5), it will use a P2 ComParam value of 20 ms.

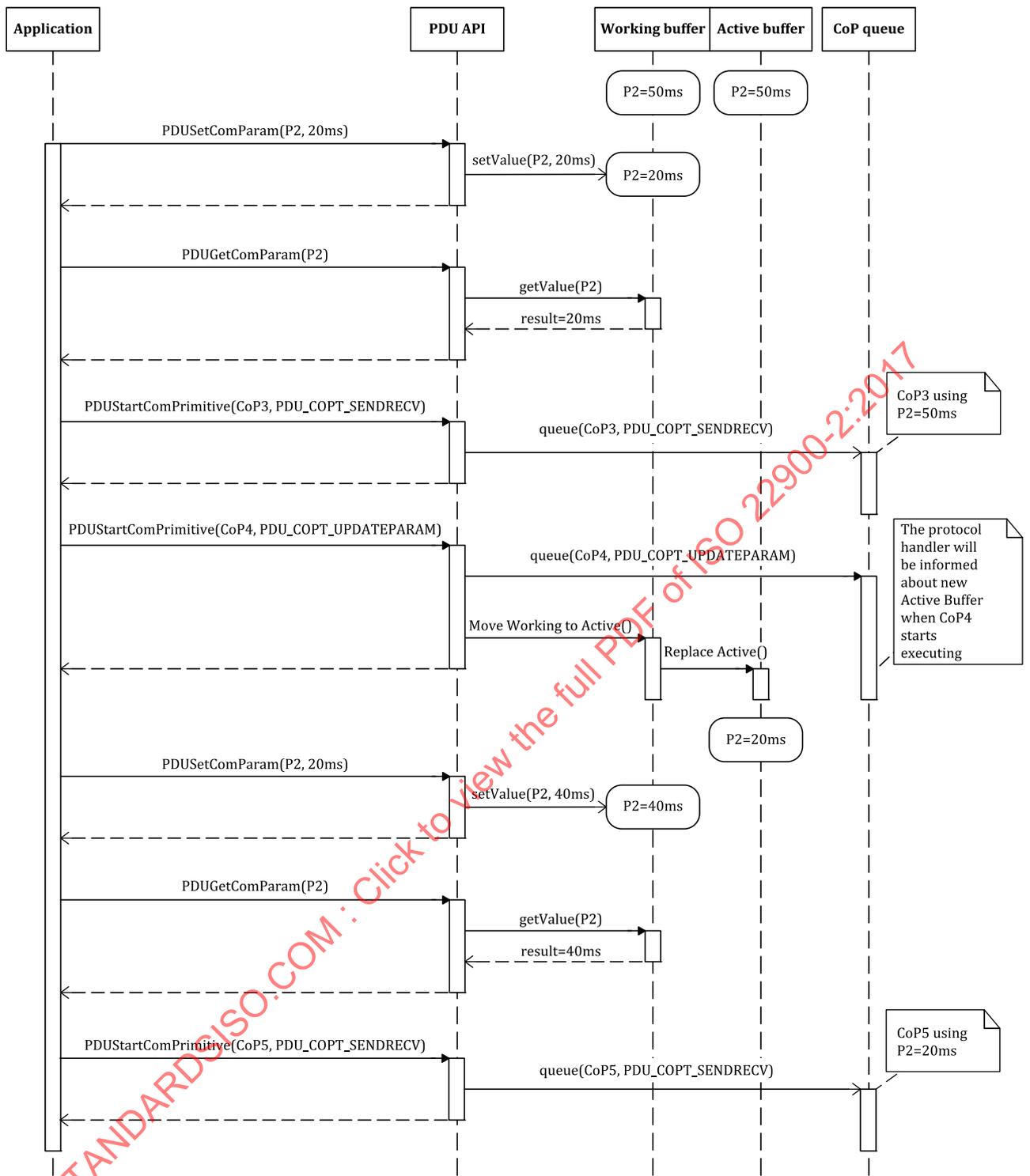


Figure 21 — Example for ComParam buffer operation using PDU\_COPT\_UPDATEPARAM (3 of 5)

8.4.15.10 Example for ComParam buffer attachment to a ComPrimitive

Figure 22 shows that each ComPrimitive gets its own ComParam buffer set at the time the ComPrimitive is placed in the CoP queue (i.e. at the time of the PDUStartComPrimitive function call). A ComParam buffer is “tied” to a ComPrimitive of types PDU\_COPT\_SENDRECV, PDU\_COPT\_STARTCOMM and PDU\_COPT\_STOPCOMM for the life of the ComPrimitive (i.e. PDU\_COPST\_FINISHED or PDU\_COPST\_CANCELLED). A queued active buffer is temporarily used for a ComPrimitive of type PDU\_COPT\_UPDATEPARAM until the ComPrimitive goes to executing (PDU\_COPST\_EXECUTING). At the time the ComPrimitive goes to executing the queued, active buffer is permanently moved to the active buffer used by the ComLogicalLink and ComPrimitives.

The physical ComParams are handled uniquely such that all ComLogicalLinks sharing a physical resource will read the same physical ComParam values. Any ComLogicalLink may modify the physical ComParams unless a ComLogicalLink has requested exclusive privilege to control the physical ComParams (see 8.4.13).

NOTE Physical ComParams cannot be changed using the TempParamUpdate flag in a PDUStartComPrimitive function call.

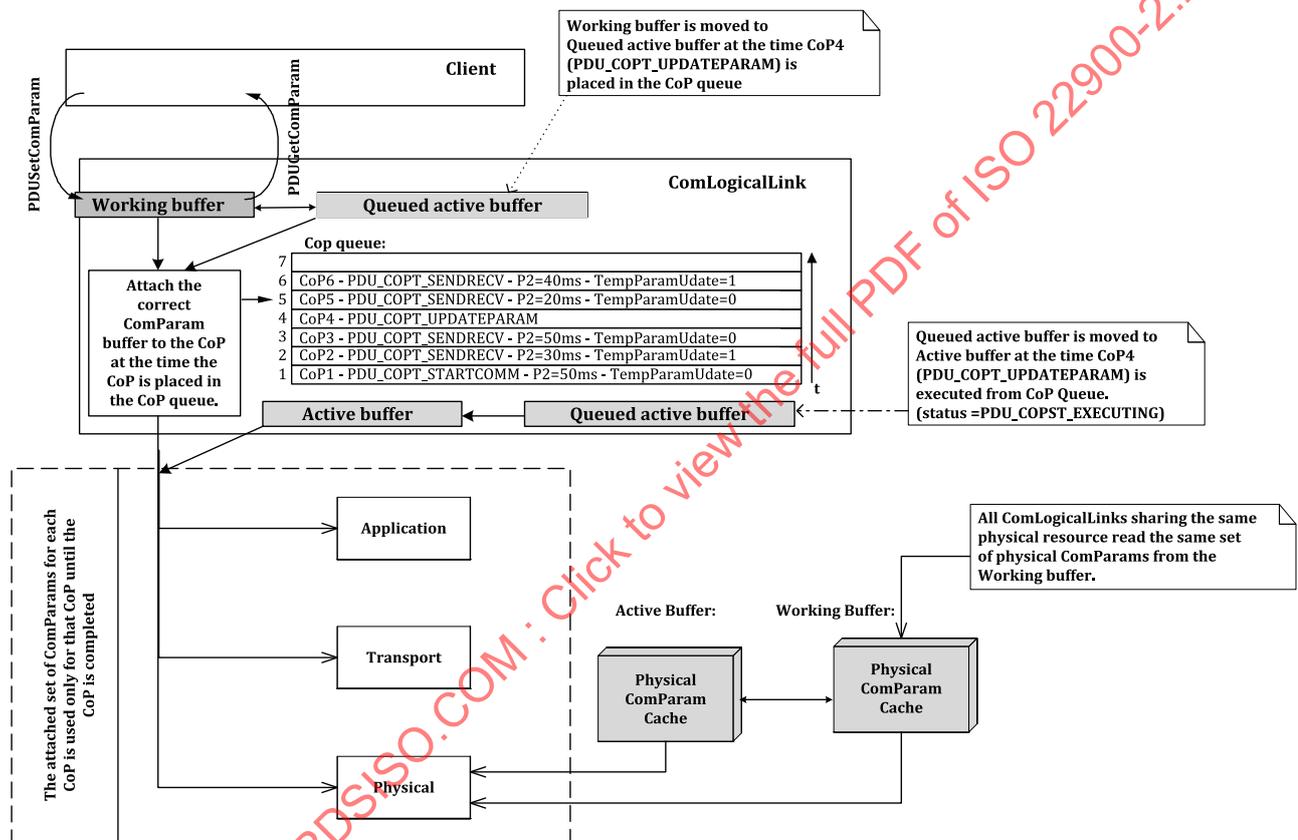


Figure 22 – Example of buffer diagram for ComParam buffer operation example (4 of 5)

### 8.4.15.11 Additional example for ComPrimitive queuing

Figure 23 gives an additional example of the order of ComPrimitive queuing for the above ComPrimitive sequence operation.

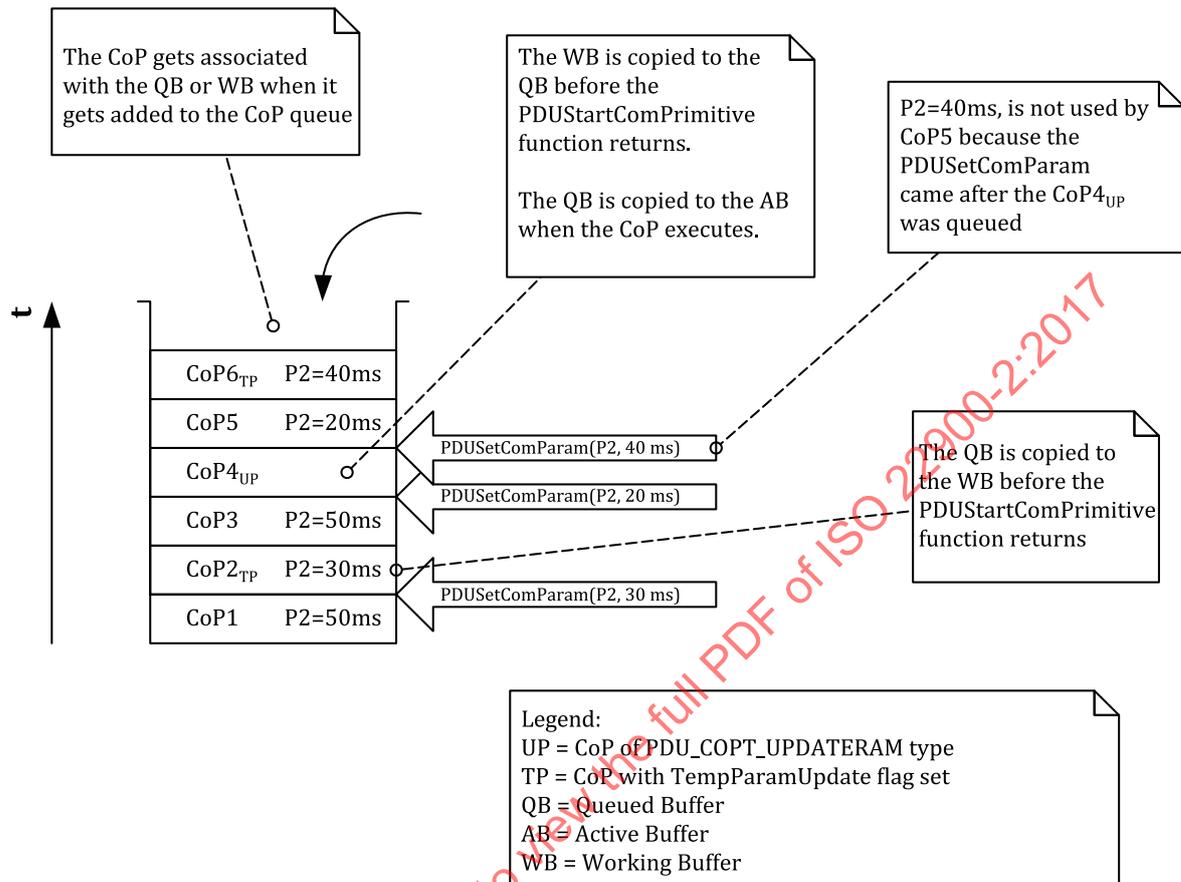


Figure 23 — Example diagram for ComPrimitive Queuing (5 of 5)

### 8.4.16 PDUSetComParam

#### 8.4.16.1 Purpose

The PDUSetComParam function transfers a ComParam setting to the D-PDU API for the given ComLogicalLink. The ComParam will be stored in a working buffer ComParam set. Thus, multiple ComParam changes can be achieved by multiple sequential calls of the PDUSetComParam function. The working buffer ComParam set of all ComParam changes will become active for the ComLogicalLink on a PDUConnect or when a ComPrimitive of type PDU\_COPT\_UPDATEPARAM is issued. A temporary set of ComParam changes can also be used for individual ComPrimitives (see 8.4.17).

#### 8.4.16.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Copy the parameter data to the ComParam working buffer.

##### 8.4.16.2.1 Behaviour — Use cases

- a) Use case: "ComLogicalLink, not connected"

The ComParam working buffer will be moved to the ComParam active buffer after calling the function PDUConnect.

## b) Use case: “ComLogicalLink, connected”

Initiating a PDUStartComPrimitive function of type PDU\_COPT\_UPDATEPARAM will copy the working buffer to the queued active buffer and queue the ComPrimitive in the ComLogicalLink’s internal ComPrimitive queue. The ComParam queued active buffer will be moved to the ComLogicalLink’s active buffer when the ComPrimitive starts execution (i.e. the ComPrimitive PDU\_COPST\_EXECUTING event occurs).

NOTE 1 If the CLL is in the PDU\_CLLST\_COMM\_STARTED state and tester present handling is enabled (see CP\_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.

NOTE 2 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP\_P3Min, CP\_P3Func and CP\_P3Phys.

## c) Use case: “ComLogicalLink, connected” and ComPrimitive (with TempParamUpdate-Flag set to '1')

This ComPrimitive will use the ComParams contained in the working buffer, NOT the active buffer. These ComParams shall be in effect for the ComPrimitive until it is finished. The ComParams for the ComPrimitive will not change even if the “active” or “working” buffers are modified by any subsequent calls to PDUSetComParam.

NOTE 3 The ComParam working buffer is restored to the active buffer when the PDUStartComPrimitive function call returns.

NOTE 4 Physical ComParams cannot be changed using the TempParamUpdate Flag.

## d) Use case: Physical BusType ComParam Change A ComParam of type ComParamType = PDU\_PC\_BUSTYPE is a physical type of ComParam.

There is only one set of physical ComParams per each physical resource (therefore, they cannot be changed temporarily using the TempParamUpdate Flag). The default behaviour is that any ComLogicalLink which shares the physical resource may modify the physical ComParams. The ComLogicalLink which uses the PDUUnlockResource to lock the physical ComParams will get exclusive rights to modify the physical ComParams for the resource (see Table 22). If a “non-owning” ComLogicalLink attempts to modify a physical ComParam, after a PDU\_COPT\_UPDATEPARAM, the D-PDU API will generate an error event item (PDU\_ERR\_EVT\_RSC\_LOCKED) indicating the requested change is not possible. If one or more ComParams cannot be set, PDU\_COPT\_UPDATEPARAM CoP generates ONE PDU\_ERR\_EVT\_RSC\_LOCKED error event. The other ComParams are set. The CLL generates a PDU\_COPST\_FINISHED event, indicating that the CoP completed. The CLL continues with the next CoPs. (The CoPs already queued at the moment the error occurs are not cancelled.)

## e) Use case: Unique ID Type ComParam Change Changes to any ComParam of type PDU\_PC\_UNIQUE\_ID are prohibited using PDUSetComParam.

A ComParam of type PDU\_PC\_UNIQUE\_ID is used only by the functions PDUSetUniqueRespIdTable and PDUGetUniqueRespIdTable.

## f) Use case: Tester Present (PDU\_PC\_TESTER\_PRESENT) Type ComParam Change Changes to any ComParam of type PDU\_PC\_TESTER\_PRESENT cannot be changed temporarily using the TempParamUpdate flag in the PDU\_COP\_CTRL\_DATA structure. (See 10.1.4.17). Once tester present handling is enabled the message is sent immediately, prior to the initial tester present cyclic time (CP\_TesterPresentTime). After initial transmission, the periodic tester present interval begins.

**8.4.16.3 C/C++ prototype**

```
EXTERNC T_PDU_ERROR PDUSetComParam(UNUM32 hMod, UNUM32 hCLL, PDU_PARAM_ITEM *pParamItem)
```

**8.4.16.4 Parameters**

hMod	Handle of MCVI protocol module.
hCLL	Handle of ComLogicalLink for which the given ComParam is to be set.

pParamItem ComParam item structure with the ComParam element to be set. The structure can be allocated from the D-PDU API by calling the function PDUGetComParam(). It has to be filled with the desired ComParam value by the application before calling this function. The value information (min value, max value, default value) can be extracted from the MDF by the application. The data structure is described in 10.1.4.5.

If the application is using the structure allocated by the D-PDU API from the PDUGetComParam() function, it is only allowed to alter the value of the data in this structure and should not increase or decrease the data length.

Table 25 specifies specific return values.

#### 8.4.16.5 Return values

**Table 25 — PDUSetComParam return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_COMPARAM_NOT_SUPPORTED	ComParam is not supported, e.g. because it is of type PDU_PC_UNIQUE_ID.
PDU_ERR_INVALID_PARAMETERS	One of the following conditions is invalid: <ul style="list-style-type: none"> <li>— Invalid ComParam ID;</li> <li>— Invalid ComParam structure;</li> <li>— NULL pointer for pParamItem.</li> </ul> ComParam value specified cannot be supported by the MVCI protocol module hardware/software.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

#### 8.4.17 PDUStartComPrimitive

##### 8.4.17.1 Purpose

The PDUStartComPrimitive function creates a ComPrimitive (used for sending/receiving data) of a given type and initiates its execution. The execution depends on the ComPrimitive type and the protocol implementation. The D-PDU data to be sent is referenced by pCoPData. The PDU\_COP\_CTRL\_DATA structure provides additional control over the execution (see 10.1.4.17). The phCoP is a call-by-reference place where the ComPrimitive handle (assigned by the D-PDU API) will be stored for further API function calls. The ComPrimitive's status (see D.1.4 and 10.1.4.11.1 for further descriptions) can be requested via the API function PDUGetStatus() or can be retrieved as an event item.

**NOTE** The D-PDU API will destroy a ComPrimitive internally as soon as the ComPrimitive has reached the status PDU\_COPST\_FINISHED or PDU\_COPST\_CANCELLED. Once a ComPrimitive has reached the FINISHED or CANCELLED state, no more result items will be queued for the ComPrimitive in the ComLogicalLink's event queue. After internal destruction of a ComPrimitive, no more operations can be executed related to this ComPrimitive. (A PDU ERROR PDU\_ERR\_INVALID\_HANDLE will be returned for a function call referencing the destroyed ComPrimitive.)

##### 8.4.17.2 Behaviour

a) Validate all input parameters.

Required pointer parameters shall not be NULL. Non-required parameters shall not be checked for NULL.

- b) Check the state of the resource used by the ComLogicalLink and return an error if it is currently unavailable (i.e. reserved by another ComLogicalLink).
- c) Place the ComPrimitive into the CoP Queue and “tie” the correct set of ComParams to the ComPrimitive.
- The ComPrimitive shall use the ComParams from the “active” buffer if TempParamUpdate is set to 0.
 

NOTE 1 At the time the ComPrimitive is queued, the set of ComParams “tied” to the ComPrimitive will be a copy of the queued active buffer. The queued active buffer will be an exact match to the active buffer for the ComLogicalLink if there is no ComPrimitive previously queued of type PDU\_COPT\_UPDATEPARAM. This ensures that the set of active ComParams for a ComPrimitive is deterministic at the time the ComParam buffer is “tied” to the ComPrimitive. See Figure 23 for an example on handling of ComPrimitive queuing.
  - The ComPrimitive shall use the ComParams from the “working” buffer if TempParamUpdate is set to 1 (see PDU\_COP\_CTRL\_DATA structure in 10.1.4.17).
 

NOTE 2 If TempParamUpdate is set to 1, the ComParam working buffer is restored to the active buffer when this PDUStartComPrimitive function call returns.

NOTE 3 Physical ComParams (ComParamType = PDU\_PC\_BUSTYPE) are not affected by the TempParamUpdate flag. Only one set of physical ComParams exist for a physical resource and they cannot be changed temporarily by a ComPrimitive. If the physical ComParams are locked by another ComLogicalLink, then a PDU\_COPT\_UPDATEPARAM will generate an error event (PDU\_ERR\_EVT\_RSC\_LOCKED) if physical ComParams are to be modified.
  - The set of ComParams “tied” to a ComPrimitive shall be in effect for the ComPrimitive until it is finished or cancelled. The ComParams for the ComPrimitive will not change even if the “active” or “working” buffer is modified by any subsequent calls to PDUSetComParam or PDUStartComPrimitive of type PDU\_COPT\_UPDATEPARAM.
  - The ComPrimitive shall use the UniqueRespIdTable from the “active” table. Temporary UniqueRespIdTables are not supported. The UniqueRespIdTable shall be in effect for the ComPrimitive until it is finished. The UniqueRespIdTable for the ComPrimitive will not change even if the “active” table is modified by any subsequent calls to PDUSetUniqueRespIdTable or PDUStartComPrimitive of type PDU\_COPT\_UPDATEPARAM.
- d) Set status of ComPrimitive to PDU\_COPST\_IDLE as it is placed in the queue.

#### 8.4.17.2.1 Behaviour — Use cases

- a) See 8.2.6 for detailed description on the different ComPrimitive types and how they are used in the different states of the ComLogicalLink.
- b) Use case: Initial receive handling

A transport layer should use the UniqueRespIdentifier table and ComParams from the currently active SendRecv ComPrimitive for initial receive handling of frames/messages. If the ComLogicalLink does not have an active SendRecv ComPrimitive, then the current active ComParam buffer should be used. Once the frame/message is bound to a ComPrimitive, the set of ComParams attached to the CoP should be used for any further processing (e.g. receive timing).

- c) Use case: CLL State = PDU\_CLLST\_OFFLINE

This is the initial state of the ComLogicalLink on creation (PDUCreateComLogicalLink) and when the ComLogicalLink has been disconnected from the vehicle bus (PDUDisconnect or on loss of communication to a module). No ComPrimitives can be added to the CLL queue while in this state (result = PDU\_ERR\_CLL\_NOT\_CONNECTED). The ComLogicalLink shall be in the state PDU\_CLLST\_ONLINE to allow any ComPrimitive queuing (see 8.4.11).

- d) Use case: CLL State Change = (any state -> PDU\_CLLST\_OFFLINE)

The ComLogicalLink transitions to PDU\_CLLST\_OFFLINE from any other ComLogicalLink state on a successful function call to PDUDisconnect or on a loss of communication to a module. All ComPrimitives currently executing (i.e. periodic) and all ComPrimitives in the CoP queue will be cancelled. A status event item, PDU\_COPST\_CANCELLED, is generated for each active CoP for the ComLogicalLink. The orders of events

under the case of losing communications to a module are: PDU\_ERR\_EVT\_LOST\_COMM\_TO\_VCI, PDU\_COPST\_CANCELLED, PDU\_CLLST\_OFFLINE and PDU\_MODST\_NOT\_AVAIL.

- e) Use case: CLL State Change = (PDU\_CLLST\_OFFLINE -> PDU\_CLLST\_ONLINE)

The ComLogicalLink changes state from PDU\_CLLST\_OFFLINE to PDU\_CLLST\_ONLINE after a successful call to PDUConnect. Some vehicle protocols require an initialization sequence (e.g. ISO 14230). Therefore, for those protocols, the ComLogicalLink shall be in the state PDU\_CLLST\_COM\_STARTED to allow for regular transmits on the vehicle bus [i.e. ComPrimitives of type PDU\_COPT\_SENDRECV with NumSendCycles != 0 will not be allowed (result = PDU\_ERR\_CLL\_NOT\_STARTED)]. Receive only ComPrimitives can be used to monitor the vehicle bus in this ComLogicalLink state (i.e. PDU\_COPT\_SENDRECV with NumSendCycles = 0 and NumReceiveCycles != 0).

- f) Use case: CLL State Change = (PDU\_CLLST\_ONLINE -> PDU\_CLLST\_COM\_STARTED)

The ComLogicalLink changes state from PDU\_CLLST\_ONLINE to PDU\_CLLST\_COM\_STARTED after successful execution of the ComPrimitive of type PDU\_COPT\_STARTCOMM. If tester present handling is enabled (see CP\_TesterPresentHandling), the message is sent immediately, prior to the initial tester present cyclic time (CP\_TesterPresentTime). After initial transmission, the tester present periodic interval is started.

NOTE Tester Present messages are only enabled in the state PDU\_CLLST\_COM\_STARTED.

- g) Use case: CLL State Change = (PDU\_CLLST\_COM\_STARTED -> PDU\_CLLST\_ONLINE)

The ComLogicalLink changes state from PDU\_CLLST\_COM\_STARTED to PDU\_CLLST\_ONLINE after successful execution of the ComPrimitive of type PDU\_COPT\_STOPCOMM. All ComPrimitives currently executing (i.e. periodic) and all ComPrimitives in the CoP queue will be cancelled when this ComPrimitive successfully executes or when the ComLogicalLink transitions to PDU\_CLLST\_OFFLINE. A status event item, PDU\_COPST\_CANCELLED, is generated for each active CoP for the ComLogicalLink.

#### 8.4.17.2.2 Behaviour — ComPrimitive status events

When a ComPrimitive changes status, a status event item is generated (see D.1.4). The following list describes each status change use case.

- a) Use case: CoP State Change (IDLE -> EXECUTING)

When a ComPrimitive is removed from the CoP Queue for execution, the status of the ComPrimitive is set to PDU\_COPST\_EXECUTING.

- If the CoP is of type PDU\_COPT\_UPDATEPARAM, copy the queued active buffer to the ComLogicalLinks active buffer and immediately set the state to PDU\_COPST\_FINISHED.

NOTE 1 If the CLL is in the PDU\_CLLST\_COMM\_STARTED state and tester present handling is enabled (see CP\_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.

NOTE 2 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP\_P3Min, CP\_P3Func and CP\_P3Phys.

- If the CoP is of type PDU\_COPT\_RESTORE\_PARAM, copy the active buffer to the working buffer and immediately set the state to PDU\_COPST\_FINISHED.
- If the protocol cannot handle the length of a ComPrimitive, an error event, PDU\_ERR\_EVT\_PROT\_ERR, is generated and the ComPrimitive is put into the FINISHED state. A protocol handler may be defined by ComParams which are used to validate a ComPrimitive size and therefore could reject a ComPrimitive based on the length of the PDU (e.g. see CP\_HeaderFormatKW).

- b) Use case: CoP State Change (EXECUTING -> WAITING)

A periodic send ComPrimitive will transition to PDU\_COPST\_WAITING after it has finished each of its periodic cycles.

c) Use case: CoP State Change (WAITING -> EXECUTING)

A periodic send ComPrimitive will transition to PDU\_COPST\_EXECUTING when it is time to begin its next transmission cycle.

d) Use case: CoP State Change (EXECUTING -> FINISHED)

A ComPrimitive will transition to PDU\_COPST\_FINISHED after it has completed execution. A periodic send ComPrimitive will transition to FINISHED after its last send cycle (NumSendCycles > 1 but not infinite (-1)). A ComPrimitive will transition to FINISHED whether it has completed successfully (e.g. all expected responses received) or unsuccessfully (e.g. receive timeout with no expected responses received).

e) Use case: CoP State Change (any state -> CANCELLED)

A ComPrimitive will transition to PDU\_COPST\_CANCELLED on the following conditions:

- a PDUDisconnect was issued for the ComLogicalLink;
- a PDUDestroyComLogicalLink was issued for the ComLogicalLink;
- a PDUcancelComPrimitive was issued for the ComPrimitive;
- a ComPrimitive of type PDU\_COPT\_STOPCOMM has completed and there were ComPrimitives currently executing or in the CoP queue;
- communications were lost to the MVCI protocol module.

8.4.17.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUStartComPrimitive(UNUM32 hMod, UNUM32 hCLL, T_PDU_COPT CoPType,
UNUM32 CoPDataSize, UNUM8 *pCoPData, PDU_COP_CTRL_DATA *pCopCtrlData, void *pCoPtag, UNUM32
*phCoP)
```

8.4.17.4 Parameters

hMod	Handle of MVCI protocol module.
hCLL	Handle of the ComLogicalLink for which the ComPrimitive shall be started.
CoPType	Type of the ComPrimitive to be started. See D.1.2 for a list of ComPrimitive types.
CoPDataSize	Size of data for the ComPrimitive; if 0, no data is supplied.
pCoPData	Reference of the buffer holding the data; NULL if no data is supplied.
pCoPCtrlData	Pointer to the control data structure for the ComPrimitive; NULL if no control data are supplied. Data structure described in 10.1.4.17.
	NOTE The PDU_COP_CTRL_DATA structure is not used for the ComPrimitives of type PDU_COPT_UPDATEPARAM and PDU_COPT_RESTORE_PARAM.
pCoPtag	Application-specific tag value providing additional information concerning the event source (e.g. pointer into application-specific structure for ComLogicalLink). The D-PDU API does not interpret this value, it is tied to the ComPrimitive and made available to the application when event items are taken from the event queue. For more information, see Annex E.
phCoP	Call-by-reference place for storage of ComPrimitive handle. This unique handle is assigned by the D-PDU API for this new ComPrimitive.

Table 26 specifies specific return values.

### 8.4.17.5 Return values

**Table 26 — PDUStartComPrimitive return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_CLL_NOT_CONNECTED	ComLogicalLink is not connected.
PDU_ERR_CLL_NOT_STARTED	Communications are not started on the ComLogicalLink yet. A Send ComPrimitive cannot be accepted in this state.
PDU_ERR_TX_QUEUE_FULL	The ComLogicalLink's transmit queue is full; the ComPrimitive could not be queued.
PDU_ERR_RSC_LOCKED_BY_OTHER_CLL	The ComLogicalLink's resource is currently locked by another ComLogicalLink.
PDU_ERR_INVALID_PARAMETERS	Invalid NULL pointer for phCoP or pCopData or pCopCtrlData or the expected response structure for a ComPrimitive with the NumReceiveCycles != 0 is NULL or has 0 entries. For ComPrimitive of type PDU_COPT_STARTCOMM or PDU_COPT_STOPCOMM or PDU_COPT_SENDRECV, NumSendCycles>0 and CoPDataSize is zero.
PDU_ERR_TEMPPARAM_NOT_ALLOWED	Physical ComParams cannot be changed as a temporary ComParam.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.18 PDUCancelComPrimitive

#### 8.4.18.1 Purpose

Cancel the current running operation for the given ComPrimitive.

#### 8.4.18.2 Behaviour

- a) Validate input parameters.
- b) Remove ComPrimitive from the CoP Queue.
- c) Set status of ComPrimitive to PDU\_COPST\_CANCELLED.
- d) If the ComPrimitive is already in the PDU\_COPST\_FINISHED status, this call will return success.

#### 8.4.18.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUCancelComPrimitive(UNUM32 hMod, UNUM32 hCLL, UNUM32 hCoP)
```

#### 8.4.18.4 Parameters

- hMod     Handle of MVCI protocol module.  
hCLL     Handle of ComLogicalLink.  
hCoP     Handle of the ComPrimitive which is to be cancelled.

Table 27 specifies specific return values.

## 8.4.18.5 Return values

Table 27 — PDUCancelComPrimitive return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_CLL_NOT_CONNECTED	ComLogicalLink is not connected.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.19 PDUGetEventItem

## 8.4.19.1 Purpose

Retrieve event item data (PDU\_EVENT\_ITEM) for given event source. PDUEventItem expects a reference of an MVCI protocol module or ComLogicalLink as input parameter to identify the event source. After retrieving the event item, the application can evaluate the type of item and then access the item-specific data.

For definition of PDU\_EVENT\_ITEM and event types, see 10.1.4.11.

## 8.4.19.2 Behaviour

a) Validate all input parameters.

NOTE 1 Pointer parameters cannot be NULL.

b) Allocate memory for PDU\_EVENT\_ITEM.

c) Fill out the event item information for the specified handle (Module or CLL).

d) Remove the event item entry from the queue. Memory for the item remains allocated.

NOTE 2 The event item is allocated and managed by the D-PDU API. It is destroyed by the application after use by calling the API function PDUDestroyItem() (see 8.4.20).

## 8.4.19.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetEventItem(UNUM32 hMod, UNUM32 hCLL, PDU_EVENT_ITEM **pEventItem)
```

## 8.4.19.4 Parameters

**hMod** Handle of the MVCI protocol module for which the event item is to be retrieved. PDU\_HANDLE\_UNDEF if an item is for a system event (e.g. D-PDU API system events like PDU\_IT\_INFO).

NOTE If hMod is set to PDU\_HANDLE\_UNDEF, then the hCLL handle is ignored.

**hCLL** Handle of the ComLogicalLink for which the event item is to be retrieved; PDU\_HANDLE\_UNDEF if an item for the given MVCI protocol module is to be retrieved (e.g. MVCI protocol module events).

**pEventItem** Call-by-reference place for storing the pointer to the event item corresponding to the given event, hMod and hCLL. Returns NULL if no result item is available. Data structure is described in 10.1.4.11.

Table 28 specifies specific return values.

### 8.4.19.5 Return values

**Table 28 — PDUGetEventItem return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	Invalid NULL pointer for pEventItem.
PDU_ERR_EVENT_QUEUE_EMPTY	No more event items are available.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.20 PDUDestroyItem

#### 8.4.20.1 Purpose

Destroy the given item (works with all created items; item data type has to be casted). See D-PDU API item type values for the different type of items for the D-PDU API.

#### 8.4.20.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Validate item type to be destroyed (see 10.1.4.2 for the description of item types).  
 c) Free memory reserved by the D-PDU API.

#### 8.4.20.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUDestroyItem(PDU_ITEM *pItem)
```

#### 8.4.20.4 Parameters

pItem Pointer to the item to be destroyed. The data structure is described in 10.1.4.2.

Table 29 specifies specific return values.

#### 8.4.20.5 Return values

**Table 29 — PDUDestroyItem return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_INVALID_PARAMETERS	Invalid item type or the pItem parameter is NULL. See 10.1.4.2 for the different type of items possible.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.21 PDURegisterEventCallback

### 8.4.21.1 Purpose

Register or unregister a callback function for event notification. By default, event notification is deactivated.

### 8.4.21.2 Behaviour

a) Validate input parameter's handles.

NOTE All handles could be PDU\_HANDLE\_UNDEF, which means that it is an event registration for the System (i.e. D-PDU API).

b) Determine whether it is a register or un-register request.

c) Either add or remove the callback function pointer to the proper object (System, Module, ComLogicalLink).

d) Figure 24 shows the internal handling of events in the D-PDU API.

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

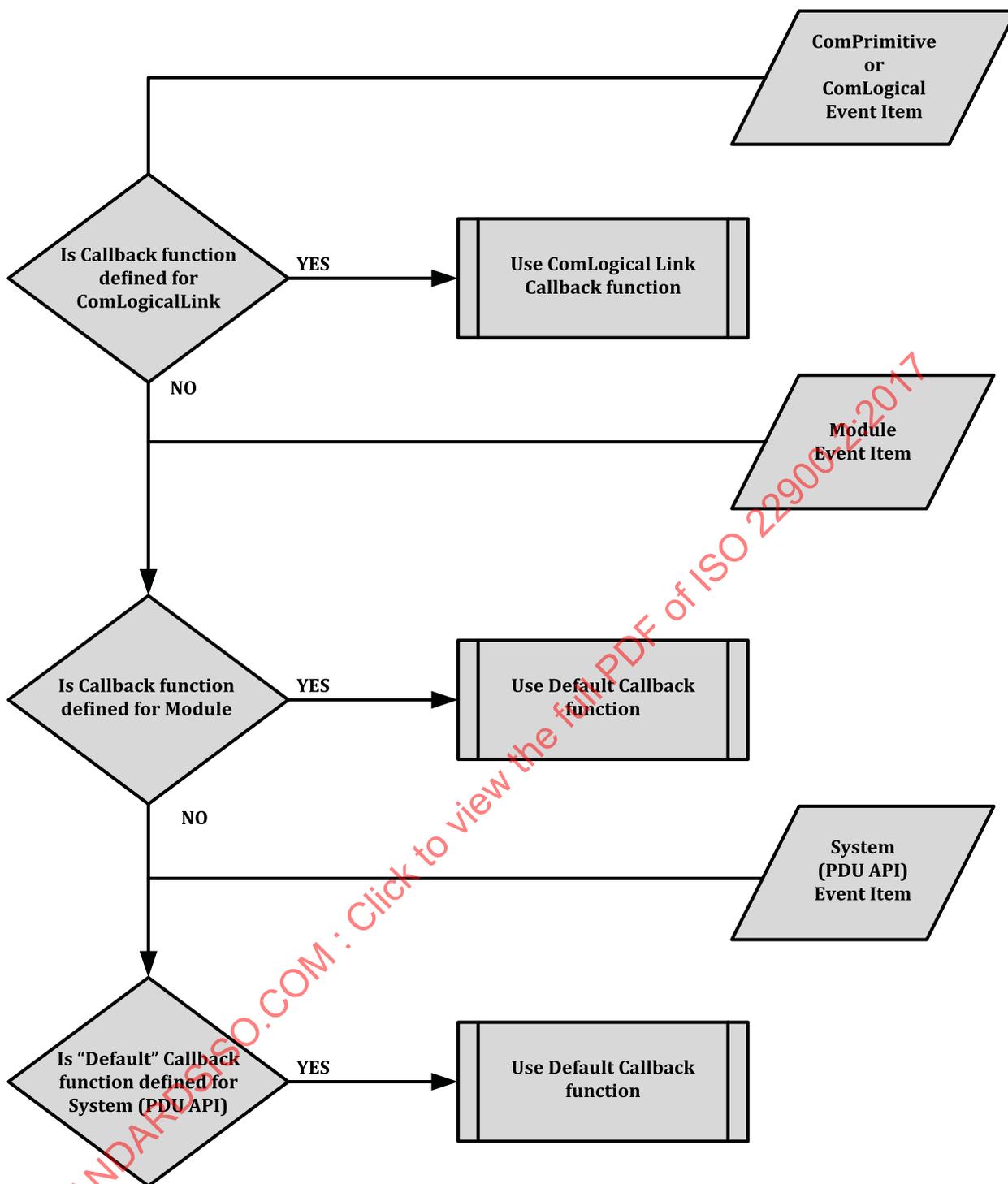


Figure 24 — Sequence of event handling

### 8.4.21.3 C/C++ Prototype

```
EXTERNC T_PDU_ERROR PDURegisterEventCallback(UNUM32 hMod, UNUM32 hCLL, CALLBACKFNC
EventCallbackFunction)
```

### 8.4.21.4 Parameters

**hMod** Handle of a Module if an event callback function shall be registered for the Module/System events. If hMod is PDU\_HANDLE\_UNDEF, the hCLL parameter is not used and the callback function is used for System event callbacks (i.e. DLL/Shared Library error events).

**hCLL** Handle of ComLogicalLink if an event callback function shall be registered with respect to a ComLogicalLink or PDU\_HANDLE\_UNDEF for registration of Module/System callback functions.

NOTE If a callback registration to a ComLogicalLink is requested after a ComLogicalLink has been connected (i.e. PDUConnect), an error will be returned.

**EventCallbackFunction** Reference of callback function to be used for event notification. Or NULL to deactivate the callback mechanism. C/C++ qualifier described in 10.1.2. Note that only one callback function can be registered for any given module or CLL. Calling PDURegisterEventCallback for a second time with same hMod and hCLL values will result in the callback function reference being updated to the newly passed reference value.

Table 30 specifies specific return values.

### 8.4.21.5 Return values

**Table 30 — PDURegisterEventCallback return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_CLL_CONNECTED	ComLogicalLink is not in the OFFLINE state and cannot accept the registration of a new callback.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.22 EventCallback prototype

#### 8.4.22.1 Purpose

The EventCallback prototype needs to be implemented and registered by the application. It is registered and known to the D-PDU API after a call of function PDURegisterEventCallback() with a reference to the application's callback function as a parameter. After registration, the application's callback function will be invoked by the D-PDU API whenever an event occurs. The callback function receives the event type, a handle of the resource (i.e. MVCI protocol module or ComLogicalLink) causing the event and an application-specific tag. The application can process the event immediately or pass it to an internal thread, which processes the events for the specific resource (i.e. MVCI protocol module or ComLogicalLink).

The runtime duration of the event callback function shall be as short as possible in order to avoid any unnecessary blocking of the D-PDU API software. The callback function will be called on the thread of the D-PDU API, therefore it is recommended that the application callback function post an event to wake another thread to do the reading of the event data. If the application shall make a D-PDU API function call in the callback routine, then PDUGetEventItem is the only permitted call.

NOTE 1 All events (status, error and results) generated by a ComLogicalLink or ComPrimitive will be placed in the ComLogicalLink's Event Queue. An event callback is either generated when there are events in the queue when the PDURegisterEventCallback function is called that registers a callback function or the ComLogicalLink's Event Queue transitions from empty to NOT empty. In other words, multiple events will not generate multiple callbacks even though each is a separate event item in the Event Queue. The application is responsible for reading ALL events from the ComLogicalLink's Event Queue before another callback will be generated.

The application shall be careful when registering the same callback function to multiple ComLogicalLinks. In this case, the callback function should be re-entrant just in case another ComLogicalLink (which may be running on a separate D-PDU API thread) makes a callback while a previous callback is currently executing.

NOTE 2 If there is NO ComLogicalLink callback registered for a specific ComLogicalLink and a Module or System callback is registered, the D-PDU API will default to using first the Module callback and otherwise use the system callback when events are received for the ComLogicalLink. The application checks the handles of the EventCallback to determine whether it is from a module, system or ComLogicalLink. This is also true if there is no module callback defined; any module events would use the system callback.

NOTE 3 In a Windows D-PDU API DLL, the callback function will have the same \_stdcall calling convention as used for all other D-PDU API functions exported from the DLL (see 9.7.2).

#### 8.4.22.2 C/C++ prototype

```
void EventCallback(T_PDU_EVT_DATA eventType, UNUM32 hMod, UNUM32 hCLL, void *pClITag, void *pAPITag)
```

#### 8.4.22.3 Parameters

eventType	Type of event which occurred (see D.1.8).
hMod	Handle of MCVI protocol module [PDU_HANDLE_UNDEF if not from a module (System event callback)].
hCLL	Handle of ComLogicalLink causing the event (PDU_HANDLE_UNDEF if not from a ComLogicalLink).
pClITag	Tag value for a ComLogicalLink. This tag should be ignored if the hCLL parameter = PDU_HANDLE_UNDEF. This is an application-specific tag value providing additional information concerning the event source (e.g. pointer onto application-specific structure for ComLogicalLink).
pAPITag	Tag value for the D-PDU API. This is an application-specific tag value providing additional information concerning the event source (e.g. pointer onto application specific structure for the D-PDU API Library).

#### 8.4.22.4 Return values

No return values are defined for this function.

### 8.4.23 PDUGetObjectid

#### 8.4.23.1 Purpose

Retrieve the item id for given item of a given type. PDUGetObjectid expects the item type and the name of the item as input parameters to identify the item. It then retrieves the id of the given item. The item id can also be obtained by parsing the MDF file.

#### 8.4.23.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Determine the id of the requested object.
- c) Fill out the response parameter pPduObjectId with the information.

### 8.4.23.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetObjectId(T_PDU_OBJT pduObjectType, CHAR8* pShortname, UNUM32
*pPduObjectId)
```

### 8.4.23.4 Parameters

pduObjectType Enumeration ID of object type. See D.1.3.

pShortname Pointer to the shortname of object, e.g. ComParam "CP\_P2Max".

pPduObjectId Call-by-reference place for storing the PDU Object ID for "Shortname" of "pduObjectType". The id will be set to PDU\_ID\_UNDEF if the D-PDU API has no valid Object Id for the requested object type and shortname.

Table 31 specifies specific return values.

### 8.4.23.5 Return values

**Table 31 — PDUGetObjectId return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (ObjectType, ShortName) or the pointer to the pPduObjectId is NULL.
PDU_ERR_MODULE_FW_OUT_OF_DATE	The D-PDU API library has a newer version than the MVCI protocol module firmware. The MVCI protocol module firmware should be updated to work with the D-PDU API Library.
PDU_ERR_API_SW_OUT_OF_DATE	The MVCI protocol module firmware has a newer version than the D-PDU API Library. The D-PDU API Library should be updated to work with the MVCI protocol module firmware.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.24 PDUGetModuleIds

#### 8.4.24.1 Purpose

Obtain module type Id, module handle information, vendor-specific string information and module status from the D-PDU API. All MVCI protocol modules detected by the D-PDU API are assigned a handle (hMod) by the D-PDU API. Each MVCI protocol module is of a certain module type (ModuleTypeId). The hMod information is used to access the individual modules in most D-PDU API function calls.

An information callback occurs (see D.1.5, PDU\_INFO\_MODULE\_LIST\_CHG) when a change in the list of MVCI protocol modules is detected by the D-PDU API. The client application should call PDUGetModuleIds again to get the new list of available MVCI protocol modules. The module handles (hMod) for modules which have already been detected will not be changed. A module which has been connected to (see 8.4.29) will maintain its handle (hMod) even after communication has been lost to the module. In this case, the module handle is destroyed only after a PDUModuleDisconnect or PDUDeconstruct.

Changes to a module connection are observed by a status change (see 10.1.4.11.1, PDU\_IT\_STATUS and D.1.4). Change of status occurs during PDUModuleConnect, PDUModuleDisconnect and loss of communications with an MVCI protocol module. Change of status does not generate a PDU\_INFO\_MODULE\_LIST\_CHG event item.

#### 8.4.24.2 Behaviour

- a) Validate input parameter.

NOTE 1 Pointer parameter cannot be NULL.

- b) Allocate PDU\_MODULE\_ITEM structure and fill the call-by-reference variable pModuleIdList. The D-PDU API structure (pModuleIdList) has to be freed by calling PDU\_DestroyItem from the application.

NOTE 2 In the case of no detection of any MVCI protocol modules, the call to PDU\_GetModuleIds will return a PDU\_MODULE\_ITEM with the number of entries set to zero (NumEntries = 0) and the pointer to PDU\_MODULE\_DATA set to NULL (pModuleData = NULL).

- c) The D-PDU API shall generate a unique handle for each MVCI protocol module interface type supported.

EXAMPLE MVCI protocol module with three interface types. The strings are vendor specific.

Table 32 shows an example of unique handles per interface type.

**Table 32 — Example of unique handles per interface type**

	Ethernet interface	Wireless interface	USB interface
hMod	0x00000001	0x00000002	0x00000003
ModuleTypeId	0x00000001	0x00000001	0x00000001
ModuleStatus	PDU_MODST_AVAIL	PDU_MODST_AVAIL	PDU_MODST_AVAIL
pVendorModuleName	"MVCI Friendly Name = 'VCI 1'"	"MVCI Friendly Name = 'VCI 1'"	"MVCI Friendly Name = 'VCI 1'"
pVendorAdditionalInfo	"Interface='Ethernet'"	"Interface='Wireless'"	"Interface='USB'"

- d) If detection of a module or module interface type is lost and the handle was in the state PDU\_MODST\_AVAIL, the handle will no longer be valid and will be removed from the list of detected modules. If the module or module interface type is re-detected a new module handle will be generated by the D-PDU API for the module. Each time the list of module handles changes, an information event will be generated to indicate that a new list of MVCI protocol module handles is available (see PDU\_INFO\_MODULE\_LIST\_CHG event).

#### 8.4.24.2.1 Behaviour — Use cases

When an MVCI protocol module changes status, a status event item is generated (see D.1.4). The following list describes each status change use case.

- a) Use case: Module State = PDU\_MODST\_AVAIL

This is the initial state of a MVCI protocol module when it is initially detected by the D-PDU API. No status event item is generated on this initial state. A module shall be in the state PDU\_MODST\_READY to allow any API function calls to the module (see 8.4.29).

- b) Use case: Module State Change = (PDU\_MODST\_AVAIL -> PDU\_MODST\_READY)

The module transitions to PDU\_MODST\_READY after a successful call to PDU\_ModuleConnect. The module is now ready to begin an API session with the client application

- c) Use case: Module State Change = (PDU\_MODST\_READY -> PDU\_MODST\_NOT\_READY)

The module transitions to PDU\_MODST\_NOT\_READY when a condition occurs on the device which prohibits execution of any further API calls. This condition may only be momentary while the module recovers from the not ready state (e.g. PDU\_IOCTL\_RESET).

- d) Use case: Module State Change = (PDU\_MODST\_READY -> PDU\_MODST\_NOT\_AVAIL) or (PDU\_MODST\_NOT\_READY -> PDU\_MODST\_NOT\_AVAIL)

The module transitions to PDU\_MODST\_NOT\_AVAIL on a loss of communication to a module. All ComPrimitives currently executing (i.e. periodic) and all ComPrimitives in the CoP queue will be cancelled (PDU\_COPST\_CANCELLED). All active ComLogicalLinks will go into the offline state (PDU\_CLLST\_OFFLINE).

The orders of events under the case of losing communications to a module are: PDU\_ERR\_EVT\_LOST\_COMM\_TO\_VCI, PDU\_COPST\_CANCELLED, PDU\_CLLST\_OFFLINE and PDU\_MODST\_NOT\_AVAIL.

e) Use case: Module State Change = (PDU\_MODST\_READY -> PDU\_MODST\_AVAIL)

The module transitions to PDU\_MODST\_AVAIL after a successful call to PDUModuleDisconnect. All resources are freed for the module.

**8.4.24.3 C/C++ prototype**

EXTERNC T\_PDU\_ERROR PDUGetModuleIds(PDU\_MODULE\_ITEM \*\*pModuleIdList)

**8.4.24.4 Parameters**

pModuleIdList Pointer for storing the pointer to Module Type Ids and the Module handles for all modules that are connected to the D-PDU API. The data structure is described in 10.1.4.6.

Table 33 specifies specific return values.

**8.4.24.5 Return values**

**Table 33 — PDUGetModuleIds return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_INVALID_PARAMETERS	Invalid NULL pointer for pModuleList.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.4.25 PDUGetResourceIds**

**8.4.25.1 Purpose**

Obtain a list of resource ids from the D-PDU API for a given module that supports the resource data information. The resource data information is defined as a protocol, bus type and pin(s). The object Ids for the resource data information can be obtained by using the PDUGetObjectId function.

The caller supplies a reference to a memory object that is of the type PDU\_RSC\_DATA. This object contains resource data information (pResourceIdData) for a single set of resource data information. The D-PDU API shall generate a PDU\_IT\_RSC\_ID object (pResourceIdList) that has a list of resource Ids that match the given resource data information. The application shall release the D-PDU API memory by calling PDUDestroyItem after it has used the resource Id list information.

**8.4.25.2 Behaviour**

a) Validate input parameters.

NOTE Pointer parameters cannot be NULL.

b) Function takes pResourceIdData structure as allocated by the application.

c) Allocate memory for the pResourceIdList result information.

d) Extracts required information from pResourceIdData structure and determines the correct list of resource Ids that match the resource data requested.

### 8.4.25.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetResourceIds(UNUM32 hMod, PDU_RSC_DATA *pResourceIdData,
PDU_RSC_ID_ITEM **pResourceIdList)
```

### 8.4.25.4 Parameters

hMod	Handle of MVCI protocol module. If set to PDU_HANDLE_UNDEF, then all modules connected to the D-PDU API will return their resource Ids and the module handles which support the PDU_RSC_DATA elements.
pResourceIdData	Call-by-reference place for the resource Id data information for a particular module type. The data structure is described in 10.1.4.8.
pResourceIdList	Call-by-reference place for storing the Resource Id list for the selected resource data. This item shall be destroyed by the application by calling PDUDestroyItem. The data structure is described in 10.1.4.7.

Table 34 specifies specific return values.

### 8.4.25.5 Return values

**Table 34 — PDUGetResourceIds return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_INVALID_PARAMETERS	The reference pointer is invalid (NULL) for pResourceIdData or pResourceIdList.
PDU_ERR_MODULE_FW_OUT_OF_DATE	The D-PDU API library has a newer version than the MVCI protocol module firmware. The MVCI protocol module firmware should be updated to work with the D-PDU API Library.
PDU_ERR_API_SW_OUT_OF_DATE	The MVCI protocol module firmware has a newer version than the D-PDU API Library. The D-PDU API Library should be updated to work with the MVCI protocol module firmware.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.26 PDUGetConflictingResources

#### 8.4.26.1 Purpose

Provide a list of resources that conflict with and cannot therefore be selected at the same time as a specified resource. The reason for the conflict may be that the resources utilize the same pin or utilize the same controller. The D-PDU API uses the MDF and CDF to extract the information from all modules and module types. It is possible to detect conflicting resources in a one-vendor D-PDU API system. When MVCI protocol modules of more than one vendor are connected by a Y-cable, the system-integrator has to take care of any conflicting resources. This information would only be addressed to multiple MVCI protocol modules if there is more than 1 MVCI protocol module connected to a vehicle. It is the responsibility of the application to determine which group of modules are connected to a single vehicle and to fill out the pInputModuleList correctly.

#### 8.4.26.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Determine all resource conflicts of ResourceId between the modules listed in pInputModuleList.
- c) Allocate memory for the PDU\_RSC\_CONFLICT\_ITEM structure.
- d) Fill the call-by-reference variable pOutputConflictList. The D-PDU API structure (pOutputConflictList) has to be freed by calling PDUDestroyItem from the application.

**8.4.26.3 C/C++ prototype**

```
EXTERNC T_PDU_ERROR PDUGetConflictingResources(UNUM32 resourceId, PDU_MODULE_ITEM
*pInputModuleList, PDU_RSC_CONFLICT_ITEM **pOutputConflictList)
```

**8.4.26.4 Parameters**

resourceId            The resource identifier to check for conflicts. The resource id is available from the MDF file and PDUGetResourceIds function.

pInputModuleList    List of modules to determine conflicts against. The data structure is described in 10.1.4.6.  
 NOTE Both hMod and ModuleType need to be valid in this structure.

pOutputConflictList Call-by-reference place for storing the information for each conflicted resource. The data structure is described in 10.1.4.9.

Table 35 specifies specific return values.

**8.4.26.5 Return values**

**Table 35 — PDUGetConflictingResources return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_PARAMETERS	Invalid resource ID or one of the reference pointers (pInputModuleList or pOutputConflictList) are invalid (NULL).
PDU_ERR_MODULE_FW_OUT_OF_DATE	The D-PDU API library has a newer version than the MVCI protocol module firmware. The MVCI protocol module firmware should be updated to work with the D-PDU API Library.
PDU_ERR_API_SW_OUT_OF_DATE	The MVCI protocol module firmware has a newer version than the D-PDU API library. The D-PDU API library should be updated to work with the MVCI protocol module firmware.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.4.27 PDUGetUniqueRespIdTable**

**8.4.27.1 Purpose**

Retrieve information of all unique response identifiers configured for the ComLogicalLink. Each unique response identifier is associated with a list of ComParams that are of type PDU\_PC\_UNIQUE\_ID.

When this function is called prior to a PDUSetUniqueRespIdTable, the structure returned will contain the ComParam information for only a single unique response and the UniqueRespIdentifier will be set to PDU\_ID\_UNDEF. The ComParam information can then be used to determine the correct set of ComParams used by the Protocol for unique ECU response identification.

Since the Unique Response ID Table is a structure holding ComParams, PDUGetUniqueRespIdTable uses the same mechanisms for handling ComParams in an internal working table as described for PDUGetComParams.

NOTE ComParams that are of type PDU\_PC\_UNIQUE\_ID can only be used with the Unique Response ID Table. They cannot be used in the functions PDUGetComParam() or PDUSetComParam().

#### 8.4.27.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Allocate PDU\_UNIQUE\_RESP\_ID\_TABLE\_ITEM structure. If the table has not been previously set by PDUSetUniqueRespIdTable, then only 1 entry will be allocated and the UniqueRespIdentifier will be PDU\_ID\_UNDEF.
- c) Fill in the table structure for the ComLogicalLink. The elements in the tables are based on the selected protocol for the ComLogicalLink. The number of ComParams in the list will be protocol dependent. The number of entries in the table can be 1 or more.

#### 8.4.27.3 C/C++ prototype

```
EXTERNC      T_PDU_ERROR      PDUGetUniqueRespIdTable(UNUM32      hMod,      UNUM32      hCLL,
PDU_UNIQUE_RESP_ID_TABLE_ITEM **pUniqueRespIdTable)
```

#### 8.4.27.4 Parameters

hMod Handle of VCI Module.

hCLL Handle of ComLogicalLink.

pUniqueRespIdTable Call-by-reference place for storing the Unique Response ID Table for the CLL; the item is allocated by the D-PDU API and has to be released after use from the application by calling the function PDUDestroyItem(). The data structure is described in 10.1.4.10.

Table 36 specifies specific return values.

#### 8.4.27.5 Return values

Table 36 — PDUGetUniqueRespIdTable return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	The pointer pUniqueRespIdTable is invalid (NULL).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.4.28 PDUSetUniqueRespIdTable

#### 8.4.28.1 Purpose

Set Unique Response Id Table information for a ComLogicalLink. This function is used to set up a table of unique response identifiers. Each unique response identifier contains a set of ComParams that uniquely define any response from a specific ECU (functional or physical responses). The application assigns the UniqueRespIdentifier. The valid range for Unique Response Identifier values is 0x0 - 0xFFFFFFFF.

The Unique Response Id Table is used for physical responses, as well as for functional responses and monitored messages. All addressing type modes (functional/physical) are contained in the list of ComParams so that any message from a specific ECU is tied to a unique ECU identifier. This allows the application to use the UniqueRespIdentifier to an ECU variant without having to interpret any protocol-specific information (e.g. CAN Ids and ECU Source Addresses).

Since the Unique Response ID Table is a structure holding ComParams, PDUSetUniqueRespIdTable uses the same mechanisms for handling ComParams in an internal working buffer as described for PDUSetComParams. The new table will only become active upon a PDUStartComPrimitive of type PDU\_COPT\_UPDATEPARAM.

**8.4.28.2 Behaviour**

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Verify that all ComParam entries in the table are of the type PDU\_PC\_UNIQUE\_ID.
- c) Store the table for ECU Response Handling in a working buffer.

**8.4.28.2.1 Behaviour — Use cases**

- a) Use case: “ComLogicalLink, not connected”

The Unique Response Identifier working table will be moved to the active table after calling the function PDUConnect.

- b) Use case: “ComLogicalLink, connected”

Initiating a PDUStartComPrimitive function of type PDU\_COPT\_UPDATEPARAM will queue the ComPrimitive in the ComLogicalLink’s internal ComPrimitive Queue. A copy of the URID table will be stored in a queued active table when the ComPrimitive is placed on the ComPrimitive Queue. The queued active table will be used for all subsequent ComPrimitives being placed on the ComPrimitive queue. The Unique Response Identifier queued active table will be moved to the ComLogicalLinks active table when the ComPrimitive changes status to EXECUTING (PDU\_COPST\_EXECUTING). This functionality is similar to the ComParam use case described in 8.4.15.10.

**8.4.28.3 C/C++ prototype**

```
EXTERNC T_PDU_ERROR PDUSetUniqueRespIdTable (UNUM32 hMod, UNUM32 hCLL,
PDU_UNIQUE_RESP_ID_TABLE_ITEM *pUniqueRespIdTable)
```

**8.4.28.4 Parameters**

- hMod Handle of VCI Module.
- hCLL Handle of ComLogicalLink.
- pUniqueRespIdTable Call-by-reference place which contains the Unique Response ID Table for the CLL. The item is allocated by the application. The data structure is described in 10.1.4.10.

Table 37 specifies specific return values.

8.4.28.5 Return values

Table 37 — PDUSetUniqueRespIdTable return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_COMPARAM_NOT_SUPPORTED	One of the ComParams in the list is not of the type PDU_PC_UNIQUE_ID.
PDU_ERR_INVALID_PARAMETERS	The pointer pUniqueRespIdTable is invalid (NULL).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

8.4.28.6 Using the Unique Response ID Table (URID table) for physical and functional addressing

The sequence diagram demonstrates how to use the Unique Response ID Table for functional and physical addressing. See Figure 25.

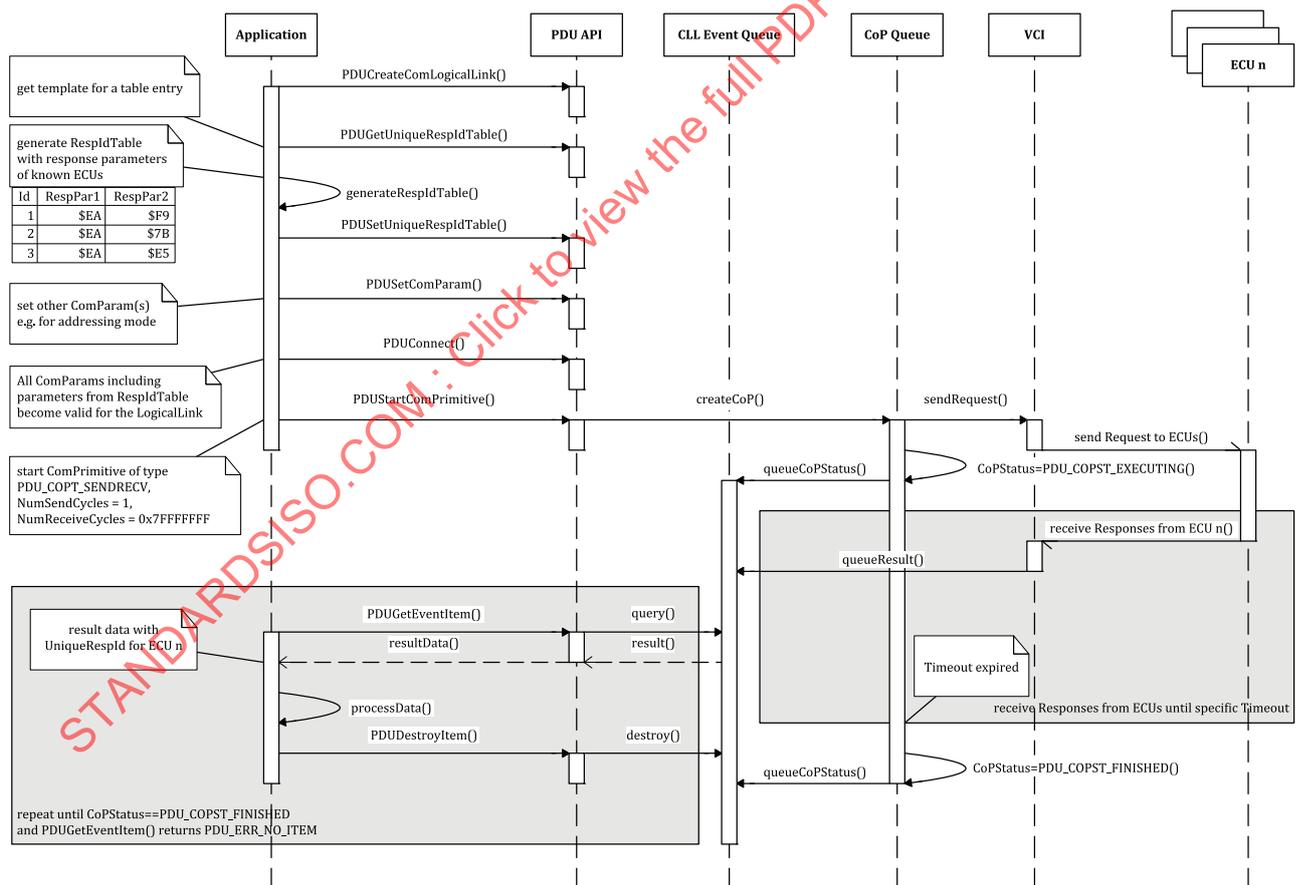


Figure 25 — Sequence of URID table for physical and functional addressing

After creating a ComLogicalLink, the application generates the Unique Response ID Table, using the template from a table entry retrieved by calling PDUGetUniqueRespIdTable(). The application does the following steps to configure the Unique Response Id Table.

- a) Add a table entry for each ECU in the functional group. If doing physical addressing, then only one entry is needed.
- b) Each entry contains a list of ComParams which uniquely identify an ECU response. These ComParams are of type PDU\_PC\_UNIQUE\_ID.
- c) Assign a Unique Response ID value for each table entry (ID range is 0 to 0xFFFFFFFF).
- d) With this generated table, the application calls PDUSetUniqueRespIdTable().
- e) The application also sets other ComParams (PDUSetComParam), e.g. to switch the addressing mode to functional addressing.

NOTE Some details are left out in the diagram, like necessary calls of PDUDestroyItem.

- f) The application calls PDUConnect(), the Unique Response ID Table and other ComParams become valid for all future ComPrimitives on the ComLogicalLink.

When a ComPrimitive is configured with NumSendCycles set to 1 and NumReceiveCycles set to -2 (IS-MULTIPLE), the MVCI protocol module expects responses from one or more ECUs. Until a specific timeout expires, the MVCI protocol module receives responses and tries to match the Unique Response ID for each response (see the following subclauses for details). The Unique Response ID is saved and then returned in a result item when the payload data is matched to a ComPrimitive expected response. The application retrieves the result items from the ComLogicalLink's Event Queue (details of event notification are not shown in the diagram). When processing the result data, the application is able to assign the data to a certain ECU via the Unique Response ID.

The sequence for physical addressing with single request/single response is very similar. The main difference is that the Unique Response ID Table contains exactly one entry, and with NumReceiveCycles set to 1, the ComPrimitive reaches the status PDU\_COPST\_FINISHED immediately when the only response is received.

#### 8.4.28.7 Handling of known and unknown responses from an ECU

##### 8.4.28.7.1 Use cases

The following use cases concerning ECU response handling have to be regarded.

- a) Use case 1: Only known responses

The application knows the response parameters of each ECU to respond to the functional/physical request. The application fills the Unique Response ID Table with an entry for each ECU. Then the D-PDU API uses the response parameters from the Unique Response ID Table entries to setup receive message acceptance filters. Thus, only responses from known ECUs with a corresponding table entry will be received.

- b) Use case 2: Only unknown responses

The application has no knowledge about the response parameters of any ECU to respond to the functional/physical request. The application uses a Unique Response ID Table with only one entry, with the UniqueRespIdentifier set to PDU\_ID\_UNDEF [response parameter values are "don't care" (i.e. NumParamItems = 0)]. Then the D-PDU API receives all possible diagnostic messages and filters them only with the expected response structure, as described below for handling of unknown ECU response Ids.

- c) Use case 3: Known and unknown responses

The application knows the response parameters of some ECUs to respond to the functional request, but there may be additional unknown ECUs responding, and the application also wants to receive these responses to detect unknown ECUs. The application fills the Unique Response ID Table with an entry for each known ECU response and an additional entry with the UniqueRespIdentifier set to PDU\_ID\_UNDEF [response parameter values are "don't care" (i.e. NumParamItems = 0)]. Then the D-PDU API receives all possible diagnostic messages and filters them only with the expected response structure. Responses with known and unknown response Ids are treated as described below.

NOTE In all cases, the receive message acceptance filters automatically configured by the D-PDU API using the Unique Response ID Table will be overridden by filters set by the application using a PDU\_IOCTL\_START\_MSG\_FILTER command.

#### 8.4.28.7.2 Handling of known ECU response ids

When the D-PDU API receives a message from an ECU, the following process steps are applied.

- a) Determine if the message passes the acceptance filters (see 8.5.14).
- b) Match the received message header information to an entry in the table of Unique Response ids. The matching algorithm is protocol specific (e.g. some protocols will use CAN ids, others will use Target Addresses, ECU Sources address, etc.).
- c) Once a UniqueRespIdentifier is found, the payload data are attempted to be matched to the ExpectedResponseStructure (see 10.1.4.19) of all active ComPrimitives (starting with the active SENDRECV ComPrimitive).
- d) When a match is found, the UniqueRespIdentifier is returned (along with the data and RxFlag information) to the application indicating which ECU the message was from (see 10.1.4.11.4).

#### 8.4.28.7.3 Handling of unknown ECU response ids

If a known ECU match cannot be found in the Unique Response Id Table and the table has an entry for unknown handling (i.e. one entry has the UniqueRespIdentifier set to PDU\_ID\_UNDEF), the following steps will be followed.

- a) Determine if the payload data can be matched to an ExpectedResponseStructure entry of an active ComPrimitive (SendRecv or RecvOnly).
- b) Set the UniqueRespIdentifier to PDU\_ID\_UNDEF in the PDU\_RESULT\_DATA for a PDU\_EVENT\_ITEM, indicating a valid message was received, but the ECU does not have a unique identifier in the URID table.
- c) Additional message header information can be obtained by setting the ENABLE\_EXTRA\_INFO bit in the TxFlag for the ComPrimitive (see D.2.1).

NOTE 1 For certain protocols, an unknown ECU response can be incomplete and can need additional flow control handling by the D-PDU API, which can be impossible without a corresponding entry in the Unique Response Id Table. Then the D-PDU API will drop this incomplete response.

NOTE 2 Depending on the protocol, it might not be possible to clearly distinguish if a response from an unknown ECU identifier or a non-diagnostic message has been received. Then the D-PDU API might not be able to deliver unexpected responses reliably.

#### 8.4.28.7.4 ECU response handling flow chart

Figure 26 shows the sequence of unique response identifiers and expected responses.

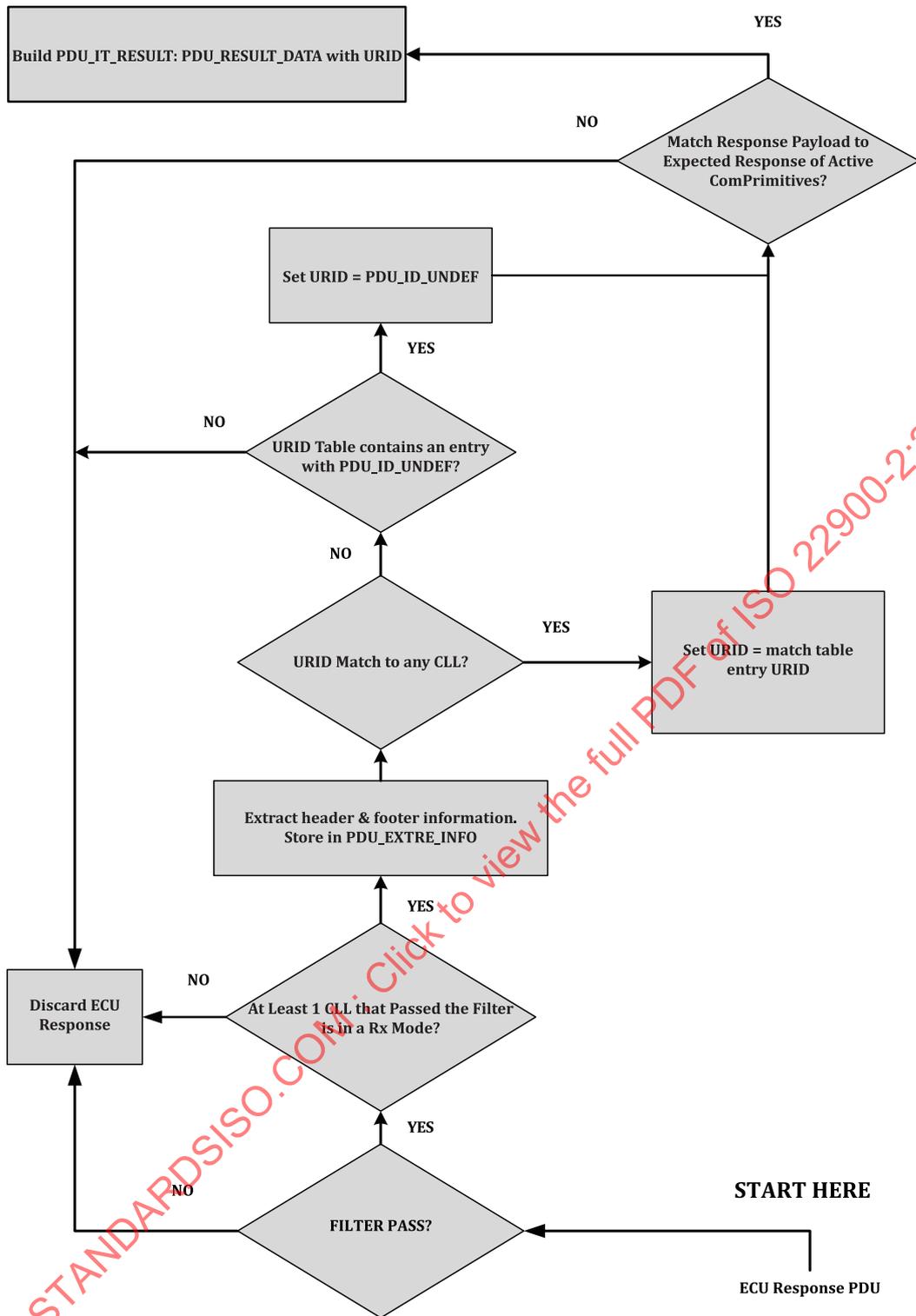


Figure 26 — Sequence of unique response identifiers and expected responses

8.4.29 PDUModuleConnect

8.4.29.1 Purpose

Establish connection to the specified MVCI protocol module and initialize its system-level drivers. Obtain available resources from the specified MVCI protocol module and create internal structures including a resource table. The communication state is offline, i.e. no allocation of resources and no communication over vehicle interface takes place.

### 8.4.29.2 Behaviour

- a) Determine if connection is available to MVCI protocol module. Module shall be in the state PDU\_MODST\_AVAIL. If connection is not possible, return error PDU\_ERR\_FCT\_FAILED.
- b) Initialize communication with the specific MVCI protocol module.
- c) Determine all resources status on the MVCI protocol module.
- d) Set the module status to PDU\_MODST\_READY.

NOTE 1 Most D-PDU API function calls, which require a hMod parameter, will return an error (PDU\_ERR\_MODULE\_NOT\_CONNECTED) if the module status is not in the state PDU\_MODST\_READY. The following list of D-PDU API functions are allowed to be used prior to a PDUModuleConnect:

- PDUGetResourceIds;
- PDUGetObjectID;
- PDUGetConflictingResources;
- PDUGetStatus;
- PDURegisterEventCallback.

NOTE 2 The following list of D-PDU API functions are allowed to be used when a module is in the state PDU\_MODST\_NOT\_AVAIL after loss of communication following a PDUModuleConnect:

- PDUGetEventItem.

NOTE 3 When the D-PDU API detects a loss of communications to an MVCI protocol module after it has been connected, the module status is set to PDU\_MODST\_NOT\_AVAIL. It is advisable that a client application calls PDUModuleDisconnect when communications have been lost to the MVCI protocol module after all items have been retrieved from the module event queue (see PDU\_ERR\_EVT\_LOST\_COMM\_TO\_VCI.)

- e) Once a module has been connected, the handle (hMod) remains valid until a PDUModuleDisconnect even after a loss of communication with the module. This behaviour is required in order to maintain the event queues for the client application retrieval of event items.

#### 8.4.29.2.1 Behaviour — Use cases

When an MVCI protocol module changes the status, a status event item is generated (see D.1.4). The following list describes each status change use case.

- a) Use case: Module State = PDU\_MODST\_AVAIL

This is the initial state of an MVCI protocol module when it is initially detected by the D-PDU API. A module shall be in the state PDU\_MODST\_READY to allow any API function calls to the module (see PDUModuleConnect).

- b) Use case: Module State Change = (PDU\_MODST\_AVAIL -> PDU\_MODST\_READY)

The module transitions to PDU\_MODST\_READY after a successful call to PDUModuleConnect. The module is now ready to begin an API session with the client application.

- c) Use case: Module State Change = (PDU\_MODST\_READY -> PDU\_MODST\_NOT\_READY)

The module transitions to PDU\_MODST\_NOT\_READY when a condition occurs on the device which prohibits execution of any further API calls. This condition may only be momentary while the module recovers from the not ready state (e.g. PDU\_IOCTL\_RESET).

- d) Use case: Module State Change = (PDU\_MODST\_READY -> PDU\_MODST\_NOT\_AVAIL) or (PDU\_MODST\_NOT\_READY -> PDU\_MODST\_NOT\_AVAIL)

The module transitions to PDU\_MODST\_NOT\_AVAIL on a loss of communication to a module. All ComPrimitives currently executing (i.e. periodic) and all ComPrimitives in the CoP queue will be cancelled (PDU\_COPST\_CANCELLED). All active ComLogicalLinks will go into the offline state (PDU\_CLLST\_OFFLINE). The orders of events under the case of losing communications to a module are: PDU\_ERR\_EVT\_LOST\_COMM\_TO\_VCI, PDU\_COPST\_CANCELLED, PDU\_CLLST\_OFFLINE and PDU\_MODST\_NOT\_AVAIL.

- e) Use case: Module State Change = (PDU\_MODST\_READY -> PDU\_MODST\_AVAIL)

The module transitions to PDU\_MODST\_AVAIL after a successful call to PDUModuleDisconnect. All resources are freed for the module.

**8.4.29.3 C/C++ prototype**

EXTERNC T\_PDU\_ERROR PDUModuleConnect (UNUM32 hMod)

**8.4.29.4 Parameters**

hMod Handle of the MVCI protocol module to be connected. If set to PDU\_HANDLE\_UNDEF, then the D-PDU API will establish a connection to all detected MVCI protocol modules. It is up to the MVCI protocol module vendor to choose which interface type of connection will be made (e.g. a vendor may choose wireless over USB if applicable).

Table 38 specifies specific return values.

**8.4.29.5 Return values**

**Table 38 — PDUModuleConnect return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_FW_OUT_OF_DATE	The D-PDU API library has a newer version than the MVCI protocol module firmware. The MVCI protocol module firmware should be updated to work with the D-PDU API library.
PDU_ERR_API_SW_OUT_OF_DATE	The MVCI protocol module firmware has a newer version than the D-PDU API library. The D-PDU API library should be updated to work with the MVCI protocol module firmware.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.4.30 PDUModuleDisconnect**

**8.4.30.1 Purpose**

Closes all open communication links and frees communication resources to the specified module. Internal memory segments shall be freed and system-level drivers disconnected. Execution of PDUModuleDisconnect does not initiate any communication on the vehicle interfaces. For a given module, after the execution of PDUModuleDisconnect, PDUModuleConnect may be called again.

**8.4.30.2 Behaviour**

- a) Close any open communication links to the specified VCI module(s).

- b) Deinitialize the specified MVCI protocol module(s).
- c) Free all internal memory associated with the MVCI protocol module(s).
- d) If communications have not been lost to the module, set the Module Status to PDU\_MODST\_AVAIL. The module handle (hMod) is still valid for further PDUModuleConnect calls.
- e) If communications to the module have been lost, then the hMod handle is no longer valid.

It is advisable that a client application calls PDUModuleDisconnect when communications have been lost to the MVCI protocol module after all items have been retrieved from the module event queue (see PDU\_ERR\_EVT\_LOST\_COMM\_TO\_VCI).

### 8.4.30.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUModuleDisconnect(UNUM32 hMod)
```

### 8.4.30.4 Parameters

hMod Handle of the MVCI protocol module to be disconnected. If set to PDU\_HANDLE\_UNDEF, then the D-PDU API will disconnect from all previously connected MVCI protocol modules.

Table 39 specifies specific return values.

### 8.4.30.5 Return values

**Table 39 — PDUModuleDisconnect return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.4.31 PDUGetTimestamp

### 8.4.31.1 Purpose

Function obtains the current time (hardware clock) from an MVCI protocol module. This time is usually derived directly from the hardware clock of the MVCI protocol module. This time is also used internally to generate the timestamps returned by PDUGetStatus and has the same unit and resolution.

### 8.4.31.2 Behaviour

- a) Validate all input parameters.

NOTE Pointer parameters cannot be NULL.

- b) Get the latest status information for the specified handle (Module) and store the information in the memory allocated by the client application.

### 8.4.31.3 C/C++ prototype

```
EXTERNC T_PDU_ERROR PDUGetTimestamp(UNUM32 hMod, UNUM32 *pTimestamp)
```

**8.4.31.4 Parameters**

hMod Handle of MvCI protocol module for which the timestamp is to be requested.  
 pTimestamp Call-by-reference place for storing timestamp in microseconds.

Table 40 specifies specific return values.

**8.4.31.5 Return values**

**Table 40 — PDUGetTimestamp return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MvCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MvCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MvCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) pTimestamp.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.5 I/O control section**

**8.5.1 IOCTL API command overview**

Table 41 gives an overview of the standard IOCTL commands for MvCI protocol modules. The following subclauses describe the details for all IOCTL commands listed in the table. In the detailed description, the input and output data, as well as the specific possible return values, are specified for each command.

- See 8.4.4 for the complete prototype of the API function.
- See 10.1.4.3 for a description of the PDU\_DATA\_ITEM structure.
- See 10.1.4.3 for a definition of the Data Item Types used for IOCTLs.

**Table 41 — Overview of PDUIoctl function — IOCTL commands**

IOCTL short name (IoctlCommandId from MDF)	Target	pInputData DataItem Type	pOutputData DataItem Type	Purpose
IOCTL short name: Short name of the specific IOCTL command from the MDF. Target: S = Command at the D-PDU API System Level M = command for MvCI protocol modules L = command for ComLogicalLinks pInputData: Description of input data for the command. pOutputData: Description of output data for the command. Purpose: Description of the command.				
PDU_IOCTL_RESET	M	—	—	Reset specific MvCI protocol module.
PDU_IOCTL_CLEAR_TX_QUEUE	L	—	—	Clear transmit queue of specific ComLogicalLink.

Table 41 — (continued)

IOCTL short name (IoCtlCommandId from MDF)	Target	pInputData DataItem Type	pOutputData DataItem Type	Purpose
IOCTL short name: Short name of the specific IOCTL command from the MDF. Target: S = Command at the D-PDU API System Level M = command for MvCI protocol modules L = command for ComLogicalLinks pInputData: Description of input data for the command. pOutputData: Description of output data for the command. Purpose: Description of the command.				
PDU_IOCTL_SUSPEND_TX_QUEUE	L	—	—	Suspend transmit queue of specific ComLogicalLink. The queue processing will be halted upon this command. This can be used to fill up a ComLogicalLink's queue with ComPrimitives to achieve a steady processing of ComPrimitives after resuming the queue (e.g. for fast flash programming operation).
PDU_IOCTL_RESUME_TX_QUEUE	L	—	—	Resume transmit queue of specific ComLogicalLink. The queue processing will be started upon this command.
PDU_IOCTL_CLEAR_RX_QUEUE	L	—	—	Clear event queue of specific ComLogicalLink.
PDU_IOCTL_READ_VBATT	M	—	PDU_IT_IO_UNUM 32	Read voltage on pin 16 of MvCI protocol module.
PDU_IOCTL_SET_PROG_VOLTAGE	M	PDU_IT_IO_PROG_VOLTAGE	—	Set the programmable voltage on the specified pin/resource of the DLC connector. The voltage and pin information are specified in the PDU_DATA_ITEM, which is passed as InputData.
PDU_IOCTL_READ_PROG_VOLTAGE	M	—	PDU_IT_IO_UNUM 32	Read the feedback of the programmable voltage.
PDU_IOCTL_GENERIC	M	PDU_IT_IO_BYTE_ARRAY	—	Allows the application to send a generic message to its drivers. The message in the data buffer is sent down to the MvCI protocol module, intercepting or interpreting it.
PDU_IOCTL_SET_BUFFER_SIZE	L	PDU_IT_IO_UNUM 32	—	Sets the buffer size limit of item, see 10.1.4.11.4.
PDU_IOCTL_START_MSG_FILTER	L	PDU_IT_IO_FILTER	—	Starts filtering of incoming messages for the specified ComLogicalLink.
PDU_IOCTL_CLEAR_MSG_FILTER	L	—	—	Clears all message filters for the ComLogicalLink.
PDU_IOCTL_STOP_MSG_FILTER	L	PDU_IT_IO_UNUM 32	—	Stops the specified filter, based on filter number.
PDU_IOCTL_SET_EVENT_QUEUE_PROPERTIES	L	PDU_IT_IO_EVENT_QUEUE_PROPERTY	—	Sets the maximum size of the ComLogicalLink event queue and the queue mode.
PDU_IOCTL_GET_CABLE_ID	M	—	PDU_IT_IO_UNUM 32	Get the Cable Id of the Cable currently connected to the MvCI protocol module.
PDU_IOCTL_SEND_BREAK	L	—	—	Sends a UART Break Signal on the ComLogicalLink.

Table 41 — (continued)

IOCTL short name (IoCtlCommandId from MDF)	Target	pInputData DataItem Type	pOutputData DataItem Type	Purpose
IOCTL short name: Short name of the specific IOCTL command from the MDF. Target: S = Command at the D-PDU API System Level M = command for MSCI protocol modules L = command for ComLogicalLinks pInputData: Description of input data for the command. pOutputData: Description of output data for the command. Purpose: Description of the command.				
PDU_IOCTL_READ_IGNITION_SENSE_STATE	M	PDU_IT_IO_UNUM32	PDU_IT_IO_UNUM32	Read the ignition sense state from the specified vehicle connector pin.
PDU_IOCTL_VEHICLE_ID_REQUEST	S, M	PDU_IT_IO_VEHICLE_ID_REQUEST	—	Send a vehicle identification request over the network on the UDP control channel to identify DoIP entities (nodes or gateways).
PDU_IOCTL_SET_ETH_SWITCH_STATE	M	PDU_IT_IO_ETH_SWITCH_STATE	—	Switch the Ethernet activation PIN on DLC.
PDU_IOCTL_GET_ENTITY_STATUS	M	PDU_IT_IO_ENTITY_ADDRESS	PDU_IT_IO_ENTITY_STATUS	Retrieve the status of a DoIP entity.
PDU_IOCTL_GET_DIAGNOSTIC_POWER_MODE	M	PDU_IT_IO_ENTITY_ADDRESS	PDU_IT_IO_UNUM32	Retrieve the diagnostic power mode of a DoIP entity.
PDU_IOCTL_GET_ETH_PIN_OPTION	M	PDU_IT_IO_UNUM32	PDU_IT_IO_UNUM32	Determine the Ethernet pinout option from the Ethernet activation PIN on DLC.

For manufacturer-specific purposes, the IOCTL list can be expanded by further commands. These commands are to be listed in the MDF by their short name, following those that are described above.

### 8.5.2 PDU\_IOCTL\_RESET

The IOCTL command PDU\_IOCTL\_RESET is used to reset the MSCI protocol module with the handle, which is passed as a parameter to the PDUIoCtl() function. The command is executed synchronously (i.e. returns after completion of the reset procedure).

InputData: NULL

OutputData: NULL

NOTE 1 The reset command will cancel all activities currently being executed by the MSCI protocol module (without proper termination). All existing ComLogicalLinks will be suspended, and receive and transmit queues will be cleared. Therefore, all associated ComPrimitives and received data items will be destroyed. All existing ComLogicalLinks will be destroyed too. All hardware properties of the MSCI protocol module (e.g. programming voltage) will be reset to the default settings. After the completion of the reset command, the application will need to use the MSCI protocol module as if it were a new MSCI protocol module.

NOTE 2 The resource table (set up after the start up of the module) will not change because of a PDU\_IOCTL\_RESET. Therefore, it is not necessary to call function PDUConstruct again after the reset.

NOTE 3 The timestamp base is reset to zero.

Table 42 specifies specific return values.

Table 42 — PDU\_IOCTL\_RESET return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.3 PDU\_IOCTL\_CLEAR\_TX\_QUEUE

The IOCTL command PDU\_IOCTL\_CLEAR\_TX\_QUEUE is used to clear the transmit queue of the ComLogicalLink with the handle, which is passed as parameter to the PDUIoCtl() function. All ComPrimitive items are destroyed in the D-PDU API internally. Further function calls of the application, which refer to destroyed ComPrimitive items, will report an error.

InputData: NULL

OutputData: NULL

To avoid overlapped operation of queue processing and queue clearing, it is recommended to execute the command PDU\_IOCTL\_SUSPEND\_TX\_QUEUE before executing PDU\_IOCTL\_CLEAR\_TX\_QUEUE.

Table 43 specifies specific return values.

Table 43 — PDU\_IOCTL\_CLEAR\_TX\_QUEUE return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.4 PDU\_IOCTL\_SUSPEND\_TX\_QUEUE

The IOCTL command PDU\_IOCTL\_SUSPEND\_TX\_QUEUE is used to suspend transmit queue's processing for the ComLogicalLink with the handle being passed as parameter to the PDUIoCtl() function.

InputData: NULL

OutputData: NULL

NOTE This command can be used to fill up a ComLogicalLink's queue with ComPrimitives before executing a PDU\_IOCTL\_RESUME\_TX\_QUEUE command. Thus, a steady processing of ComPrimitives can be achieved (e.g. for fast flash programming operation).

Table 44 specifies specific return values.

Table 44 — PDU\_IOCTL\_SUSPEND\_TX\_QUEUE return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.5 PDU\_IOCTL\_RESUME\_TX\_QUEUE

The IOCTL command PDU\_IOCTL\_RESUME\_TX\_QUEUE is used to resume the transmit queue's processing for the ComLogicalLink with the handle being passed as parameter to the PDUIoctl() function.

InputData: NULL

OutputData: NULL

Table 45 specifies specific return values.

Table 45 — PDU\_IOCTL\_RESUME\_TX\_QUEUE return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.6 PDU\_IOCTL\_CLEAR\_RX\_QUEUE

The IOCTL command, PDU\_IOCTL\_CLEAR\_RX\_QUEUE, is used to clear the event queue for the appropriate ComLogicalLink (a handle is passed as an input parameter to the PDUIoctl() function). All event items (i.e. result data, information about errors or status changes) in the event queue of the ComLogicalLink will be cleared and automatically destroyed (i.e. the D-PDU API internally performs a PDUDestroyItem call for each item in the event queue).

InputData: NULL

OutputData: NULL

Table 46 specifies specific return values.

**Table 46 — PDU\_IOCTL\_CLEAR\_RX\_QUEUE return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.5.7 PDU\_IOCTL\_CLEAR\_TX\_QUEUE\_PENDING**

The IOCTL command PDU\_IOCTL\_CLEAR\_TX\_PENDING is used to clear pending transmits in the ComLogicalLink with the handle, which is passed as a parameter to the PDUIoctl() function. All pending ComPrimitive items to be transmitted (CoPs in the PDU\_COPST\_IDLE or PDU\_COPST\_WAITING states) are cancelled in the D-PDU API internally, but ComPrimitives currently in the Receive Only list, are not cancelled.

InputData: NULL

OutputData: NULL

Table 47 specifies specific return values.

**Table 47 — PDU\_IOCTL\_CLEAR\_TX\_QUEUE\_PENDING return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.5.8 PDU\_IOCTL\_READ\_VBATT**

The IOCTL command PDU\_IOCTL\_READ\_VBATT is used to read the voltage on pin 16 of the MVCI protocol module's connector. The MVCI protocol module handle is passed as a parameter to the PDUIoctl() function. The voltage will be written to the UNUM32 value (4 data bytes) of the PDU\_DATA\_ITEM structure being passed as OutputData by reference. For a description of PDU\_DATA\_ITEM, see 10.1.4.3.

InputData: NULL

OutputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_UNUM32

pData UNUM32 Vbat\_mv; /\* vehicle battery in mV \*/

Table 48 specifies specific return values.

Table 48 — PDU\_IOCTL\_READ\_VBATT return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MPCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MPCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MPCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.9 PDU\_IOCTL\_SET\_PROG\_VOLTAGE

The IOCTL command PDU\_IOCTL\_SET\_PROG\_VOLTAGE is used to set the programmable voltage on the specified pin of the DLC connector. The MPCI protocol module handle is passed as parameter to the PDUIoctl() function. The voltage and pin information are specified in the PDU\_DATA\_ITEM, which is passed as InputData. For a description of PDU\_DATA\_ITEM, see 10.1.4.3. Valid values are 5 000 mV to 20 000 mV (limited to 100 mA with a resolution of  $\pm 100$  mV). See also Table 49.

InputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_PROG\_VOLTAGE

pData pointer PDU\_IO\_PROG\_VOLTAGE\_DATA structure (see 10.1.4.3.2)

OutputData: NULL

Table 49 specifies PDU\_IOCTL programming voltage.

Table 49 — PDU\_IOCTL programming voltage description

Coded value of voltage	Meaning
0x00001388 - 0x00004E20	5 000 mV - 20 000 mV
0xFFFFFFFF	SHORT_TO_GROUND (zero impedance)
0xFFFFFFFF	VOLTAGE_OFF (high impedance)

Table 50 specifies specific return values.

Table 50 — PDU\_IOCTL\_SET\_PROG\_VOLTAGE return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MPCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MPCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MPCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_VOLTAGE_NOT_SUPPORTED	The voltage is not supported by the MPCI protocol module.

Table 50 — (continued)

Definition	Description
PDU_ERR_MUX_RSC_NOT_SUPPORTED	The specified pin/resource are not supported by the MVCI protocol module.
PDU_ERR_RESOURCE_BUSY	Resource is busy; the application has to execute the command again.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.10 PDU\_IOCTL\_READ\_PROG\_VOLTAGE

The IOCTL command PDU\_IOCTL\_READ\_PROG\_VOLTAGE is used to read the feedback of the programmable voltage from the voltage source, which is set by the command PDU\_IOCTL\_SET\_PROG\_VOLTAGE. The MVCI protocol module handle is passed as parameter to the PDUIoctl() function. The voltage will be written to the UNUM32 value (4 data bytes) of the PDU\_DATA\_ITEM structure being passed as OutputData by reference. For a description of PDU\_DATA\_ITEM, see 10.1.4.3. See also Table 49.

InputData:

OutputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_UNUM32

pData UNUM32 ProgVoltage\_mv; /\* programming voltage in mV \*/

Table 51 specifies specific return values.

Table 51 — PDU\_IOCTL\_READ\_PROG\_VOLTAGE return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.11 PDU\_IOCTL\_GENERIC

This command was added due to compatibility reasons for RP1210a. It allows the application to send a generic message to its drivers. The D-PDU API simply passes the message in the data buffer down to MVCI protocol module, if any, associated with the device hardware without intercepting or interpreting it. The generic command will be written to the element "Data" of the PDU\_DATA\_ITEM structure as a free form buffer of bytes. The PDU\_DATA\_ITEM structure is passed as InputData by reference. For a description of PDU\_DATA\_ITEM, see 10.1.4.3.

InputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_BYTEARRAY

pData pointer PDU\_IO\_BYTEARRAY\_DATA structure (see 10.1.4.3.3)

OutputData: NULL

Table 52 specifies specific return values.

**Table 52 — PDU\_IOCTL\_GENERIC return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.12 PDU\_IOCTL\_SET\_BUFFER\_SIZE

This IOCTL command sets the maximum buffer size of the received PDU on a ComLogicalLink. (See 10.1.4.11.4, PDU\_RESULT\_DATA.) The buffer size is contained in the UNUM32 value (4 data bytes) of the PDU\_DATA\_ITEM structure being passed as InputData by reference. For a description of PDU\_DATA\_ITEM, see 10.1.4.3.

InputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_UNUM32

pData UNUM32 MaxRxBufferSize; /\* maximum size of a received PDU for the ComLogicalLink \*/

OutputData: NULL

Table 53 specifies specific return values.

**Table 53 — PDU\_IOCTL\_SET\_BUFFER\_SIZE return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.13 PDU\_IOCTL\_GET\_CABLE\_ID

To let the application know which cable is currently connected to an MVCI protocol module, the following PDU\_IOCTL command can be used.

InputData: NULL

OutputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_UNUM32

pData UNUM32 CableId; /\* Cable Id from CDF \*/

With the cable ID, the application can retrieve information about the cable from the CDF, like short name, description and DLCType (connector type).

Table 54 specifies specific return values.

**Table 54 — PDU\_IOCTL\_GET\_CABLE\_ID return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MCVI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MCVI protocol module Handle or ComLogicalLink handle.
PDU_ERR_CABLE_UNKNOWN	Cable is unknown.
PDU_ERR_NO_CABLE_DETECTED	No cable is detected.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.5.14 PDU\_IOCTL\_START\_MSG\_FILTER**

This IOCTL command starts filtering incoming messages for the specified ComLogicalLink. A minimum of 64 filters can be supported per ComLogicalLink. A PDU\_DestroyComLogicalLink shall delete all its defined message filters. Filtering will only become active when the ComLogicalLink is in the PDU\_LLST\_ONLINE state (see 8.4.11). If the application does not configure any filters, the D-PDU API will automatically determine a set of filters by using the PDU\_PC\_UNIQUE\_ID ComParams configured for the ComLogicalLink (see 8.4.28). Any filters set by the application using the IOCTL filter commands will override any filters internally configured by the D-PDU API.

All Protocols:

- Pass filters and block filters will be applied to all received messages. They shall not be applied to indications or loopback messages.
- Messages that match a pass filter can still be blocked by a block filter (see Figure 27).

For the ISO 15765 protocol:

- Pass filters and block filters are applied to CAN ID filtering. They shall not be applied to indications or loopbacks of CAN IDs.

NOTE The UniqueRespIdTable (see 10.1.4.10) is used for USDT/UUDT frame handling plus flow control and extended address handling.

InputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_FILTER

pData pointer PDU\_IO\_FILTER\_LIST structure (see 10.1.4.3.4)

OutputData: NULL

Table 55 specifies specific return values.

**Table 55 — PDU\_IOCTL\_START\_MSG\_FILTER return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MCVI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MCVI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MCVI protocol module has not been connected. See PDU_ModuleConnect function.

Table 55 — (continued)

Definition	Description
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

8.5.14.1 MSG\_FILTER block diagram

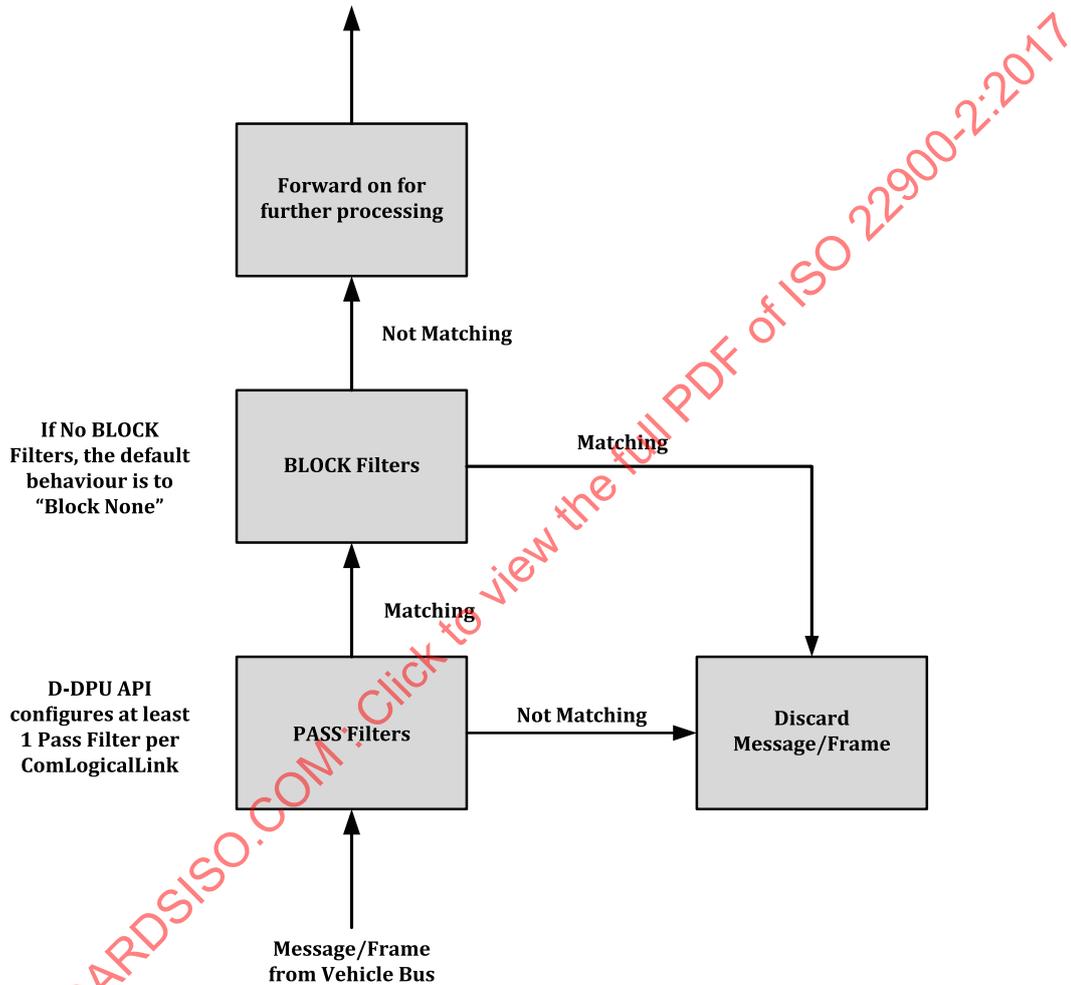


Figure 27 — MSG\_FILTER block diagram

8.5.15 PDU\_IOCTL\_STOP\_MSG\_FILTER

The IOCTL command PDU\_IOCTL\_STOP\_MSG\_FILTER removes the specified filter from the ComLogicalLink.

InputData: Value settings for PDU\_DATA\_ITEM  
 ItemType PDU\_IT\_IO\_UNUM32  
 pData UNUM32 FilterNumber; /\* Filter Number to stop \*/  
 OutputData: NULL

Table 56 specifies specific return values.

Table 56 — PDU\_IOCTL\_STOP\_MSG\_FILTER return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData) or the Filter Number is invalid.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

#### 8.5.16 PDU\_IOCTL\_CLEAR\_MSG\_FILTER

The IOCTL command PDU\_IOCTL\_CLEAR\_MSG\_FILTER removes all message filters from the ComLogicalLink.

InputData: NULL

OutputData: NULL

Table 57 specifies specific return values.

Table 57 — PDU\_IOCTL\_CLEAR\_MSG\_FILTER return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

#### 8.5.17 PDU\_IOCTL\_SET\_EVENT\_QUEUE\_PROPERTIES

The IOCTL command PDU\_IOCTL\_SET\_EVENT\_QUEUE\_PROPERTIES sets the properties of the ComLogicalLink event queue. There are two properties associated with an event queue: the event queue size and the queuing mechanism to be used. The PDU\_IOCTL\_SET\_EVENT\_QUEUE\_PROPERTIES can only be used prior to calling the PDUConnect function. If the ComLogicalLink is already connected, the function will return the PDU\_ERR\_CLL\_CONNECTED error.

The queue mode sets the behaviour of the queuing mechanism in case the ComLogicalLink reaches the maximum size of the event queue.

Table 58 defines three types of queue modes.

Table 58 — Queue mode types

Queue mode type	Description
Unlimited mode	An attempt is made to allocate memory for every item being placed on the event queue. In unlimited mode, the QueueSize is ignored. (Default mode for ComLogicalLink)
Limited mode	When the ComLogicalLink's event queue is full (i.e. maximum size has been reached), no new items are placed on the event queue. The event items are discarded in this case.
Circular mode	When the ComLogicalLink's event queue is full (i.e. maximum size has been reached), then the oldest event item in the queue is deleted so that the new event item can then be placed in the event queue.

When a ComLogicalLink reaches a queue full state, the special PDU\_EVT\_DATA\_LOST event is generated. No result items will be created (i.e. no PDU\_IT\_ERROR items will be attempted to be placed on the event queue). See D.1.8 for event types.

InputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_EVENT\_QUEUE\_PROPERTY

pData pointer PDU\_IO\_EVENT\_QUEUE\_PROPERTY\_DATA structure (see 10.1.4.3.6)

OutputData: NULL

Table 59 specifies specific return values.

Table 59 — PDU\_IOCTL\_SET\_EVENT\_QUEUE\_PROPERTIES return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MSCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MSCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MSCI protocol module has not been connected. See PDU_ModuleConnect function.
PDU_ERR_CLL_CONNECTED	CLL is already in the "online" state.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.18 PDU\_IOCTL\_SEND\_BREAK

The IOCTL command PDU\_IOCTL\_SEND\_BREAK is used to send a break signal on the ComLogicalLink. A break signal can only be sent on certain physical layers (e.g. SAE J1850 VPW physical links and UART physical links). If the link does not support the break feature, a PDU\_ERR\_FCT\_FAILED will be returned.

UART break signals are caused by sending continuous (0) values (no Start or Stop bits). The break signal shall be of a duration longer than the time it takes to send a complete byte plus Start, Stop and Parity bits. Most UARTs can distinguish between a Framing Error and a Break, but if the UART cannot do this, the Framing Error detection can be used to identify Breaks.

SAE J1850 break signals are determined by observing the timing of the active to passive transition. If the transition does not occur until after 240 µs, the current signal will be considered a valid break signal. A break signal should be followed by a SOF signal beginning with the next message to be transmitted onto the SAE J1850 bus. All nodes on a SAE J1850 bus shall return to normal operating conditions after detecting a break signal. Many SAE J1850 hardware components support the Break signal feature (transmit and receive). There is no specification on the maximum length of a SAE J1850 break signal, but it shall not be excessively long. Therefore, the maximum length shall be greater than the minimum length of 240 µs.

The ComLogicalLink's handle is passed as a parameter to the PDU\_IOCTL() function.

InputData: NULL

OutputData: NULL

Table 60 specifies specific return values.

**Table 60 — PDU\_IOCTL\_SEND\_BREAK return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_CLL_NOT_CONNECTED	ComLogicalLink is not connected.
PDU_ERR_CLL_NOT_STARTED	Communications are not started on the ComLogicalLink yet. A Send ComPrimitive cannot be accepted in this state.
PDU_ERR_RSC_LOCKED_BY_OTHER_CLL	The ComLogicalLink's resource is currently locked by another ComLogicalLink.
PDU_ERR_INVALID_PARAMETERS	Invalid (NULL) pVersionData parameter.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

#### 8.5.19 PDU\_IOCTL\_READ\_IGNITION\_SENSE\_STATE

The IOCTL command PDU\_IOCTL\_READ\_IGNITION\_SENSE\_STATE is used to read the switched vehicle battery voltage (Ignition on/off) pin. In accordance with ISO 22900-1, this information is available on pin 24 of the MVCI module chassis connector. There is no corresponding pin on the legislated ISO 15031-3/SAE J1962 compatible vehicle connectors.

This IOCTL allows for reading of a specified vehicle connector pin to determine the state of the ignition switch. Since a MVCI protocol module vendor may support a cable type which routes the ignition sense to pin 24 of the module, a DLC pin number of 0 will indicate that the switched vehicle battery voltage is to be read from the MVCI protocol module pin 24 and not from a DLC pin.

The D-PDU API will determine the sense of the ignition by first reading the permanent positive battery voltage from the vehicle [UBATvehicle (pin 16 on the DLC)] and then reading the specified switched vehicle battery voltage pin. Ignition ON will be  $\pm 2$  volts of the permanent vehicle battery voltage.

InputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_UNUM32

pData UNUM32 DLCPinNumber; /\* Pin number of the vehicles data link connector which contains the vehicle switched battery voltage. If DLCPinNumber = 0, then the ignition sense is routed to pin 24 of the MVCI protocol module\*/

OutputData: Value settings for PDU\_DATA\_ITEM

ItemType PDU\_IT\_IO\_UNUM32

pData UNUM32 IgnitionState; /\* Evaluated state of switched vehicle battery voltage.

0 = Ignition OFF  
1 = Ignition ON\*/

Table 61 specifies specific return values.

Table 61 — PDU\_IOCTL\_READ\_IGNITION\_SENSE\_STATE return values

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.5.20 PDU\_IOCTL\_VEHICLE\_ID\_REQUEST

### 8.5.20.1 General

The IOCTL command PDU\_IOCTL\_VEHICLE\_ID\_REQUEST is used to initiate a DoIP vehicle identification request. If the handle to the MVCI protocol module is undefined (hMod=PDU\_HANDLE\_UNDEF), the D-PDU API will use the operating system's network adaptors to transmit the vehicle identification request.

InputData: Value settings for PDU\_DATA\_ITEM  
 ItemType PDU\_IT\_IO\_VEHICLE\_ID\_REQUEST  
 pData pointer to PDU\_IO\_VEHICLE\_ID\_REQUEST structure (see 10.1.4.3.7)  
 OutputData: NULL

The vehicle identification request IOCTL is related to

- either a certain MVCI device (selected by a corresponding module handle as input parameter): the identification request is carried out in the network connected to the MVCI device,
- or a network adaptor which is not part of a certain MVCI device, but used by the D-PDU API implementation directly (e.g. the Ethernet adaptor of the PC system): in this case, the module handle is PDU\_HANDLE\_UNDEF).

NOTE 1 Depending on the parameter "CombinationMode", the D-PDU API assigns a DoIP entity to a single DoIP MVCI module or a combination of multiple DoIP entities to a single DoIP MVCI module.

NOTE 2 An MVCI module of type MVCI\_ISO\_13400\_DoIP\_Vehicle, MVCI\_ISO\_13400\_DoIP\_Group or MVCI\_ISO\_13400\_DoIP\_Collection requires combining only DoIP entities with different logical address.

NOTE 3 It is an expected common use case that an MVCI device is connected to only one specific vehicle using an OBD-connector including Ethernet connector pins, and also pins for other physics, e.g. CAN. In this case, only the DoIP entities of the specific vehicle will answer the vehicle identification request carried out at the MVCI device. If the application has no knowledge about the IDs (EntityID, VIN or GroupID) of the DoIP entities in the connected vehicle, the application may execute a vehicle identification request at the specific MVCI device and set CombinationMode to 3 to detect all DoIP entities of the connected vehicle and combine them in a DoIP MVCI module of type "DoIP-collection".

An MVCI device may contain resources for different protocols, except DoIP.

NOTE 4 Each DoIP MVCI module contains exactly one resource for the protocol DoIP, i.e. PDUGetResourceIds() delivers exactly one resource item.

NOTE 5 A DoIP MVCI module has limited capabilities, e.g. no PDUIoCtl() or PDUGetVersion() can be carried out.

NOTE 6 To handle combination mode in case of spontaneous vehicle announcement, use combination mode of latest Vehicle identification request or default instead (no combination).

The DoIP entities inside a DoIP MVCI module may change when new DoIP entities are announced or existing DoIP entities quit their connection. In this case, the D-PDU API shall generate an Event PDU\_INFO\_MODULE\_LIST\_CHG.

The application shall then call `PDUGetModuleIds` again to get an updated list of modules, where `pVendorAdditionalInfo` is updated with the currently available DoIP entities for each DoIP MSCI module.

NOTE 7 When a DoIP MSCI module loses connection to all contained DoIP entities, this case is handled in the same way as an MSCI module connection loss, as described in D-PDU API.

NOTE 8 The IOCTL function returns when the time defined in `VehicleDiscoveryTime` has elapsed.

After successful completion of the `PDU_IOCTL_VEHICLE_ID_REQUEST` IOCTL, the D-PDU API's list of modules will change when new DoIP MSCI modules have been discovered. An information event of type `PDU_INFO_MODULE_LIST_CHG` will be generated by the D-PDU API, upon which the application shall call `PDUGetModuleIDs()` to obtain the changed list of MSCI protocol modules (see K.2.4.3).

### 8.5.20.2 PDU\_IOCTL\_VEHICLE\_ID\_REQUEST return values

Table 62 specifies specific return values for `PDU_IOCTL_VEHICLE_ID_REQUEST`.

Table 62 — `PDU_IOCTL_VEHICLE_ID_REQUEST` return values

Definition	Description
<code>PDU_ERR_PDUAPI_NOT_CONSTRUCTED</code>	D-PDU API has not been constructed before.
<code>PDU_ERR_COMM_PC_TO_VCI_FAILED</code>	Communication between host and MSCI protocol module failed.
<code>PDU_ERR_INVALID_HANDLE</code>	Invalid MSCI protocol module Handle or <code>ComLogicalLink</code> handle.
<code>PDU_ERR_IP_PROTOCOL_NOT_SUPPORTED</code>	IP protocol is not supported: e.g. IPv6 used as <code>protocolVersion</code> , but OS does not support IPv6 (or it is disabled).
<code>PDU_ERR_ID_NOT_SUPPORTED</code>	Requested ID is not supported.
<code>PDU_ERR_INVALID_PARAMETERS</code>	At least one of the parameters is invalid ( <code>pInputData</code> and/or <code>pOutputData</code> ).
<code>PDU_ERR_FCT_FAILED</code>	Function call failed.
<code>PDU_STATUS_NOERROR</code>	Function call successful.

### 8.5.21 PDU\_IOCTL\_SET\_ETH\_SWITCH\_STATE

#### 8.5.21.1 General

The DoIP standard ISO 13400-3 provides the possibility to switch off the Ethernet chip in a DoIP entity. The IOCTL command is used to switch the sense state of the Ethernet activation pin in the DLC connector attached to an MSCI module. The pin is used to activate or deactivate the Ethernet chip in the DoIP entities which are connected to the DLC connector.

Upon successful completion of this IOCTL, the DoIP communication lines will be connected on the MSCI diagnostic connector. The actual pins connected will be selected based on the identified DoIP variant (by default this will be pins 3, 11, 12 and 13, but may be updated by making use of the IOCTL `PDU_IOCTL_GET_ETH_PIN_OPTION`; see 8.5.24).

InputData: Value settings for `PDU_DATA_ITEM`

ItemType: `PDU_IT_IO_ETH_SWITCH_STATE`

pData: pointer to `PDU_IO_ETH_SWITCH_STATE` structure (see 10.1.4.3.8)

OutputData: NULL

NOTE 1 When a DoIP entity uses the Ethernet activation pin, no DoIP communication is possible before switching on the Ethernet activation.

NOTE 2 After switching on the Ethernet activation, the DoIP entity sends out a Vehicle Announcement Message called VAM.

NOTE 3 This IOCTL is only related to MSCI use case 1 (with MSCI device). See K.2.4.

NOTE 4 ISO 13400-4 defines the Ethernet activation pin 8.

### 8.5.21.2 PDU\_IOCTL\_SET\_ETH\_SWITCH\_STATE return values

Table 63 specifies specific return values for PDU\_IOCTL\_SET\_ETH\_SWITCH\_STATE.

**Table 63 — PDU\_IOCTL\_SET\_ETH\_SWITCH\_STATE return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

### 8.5.22 PDU\_IOCTL\_GET\_ENTITY\_STATUS

#### 8.5.22.1 General

Retrieves the status of a DoIP entity. This IOCTL should be executed on the DoIP module the specific DoIP entity belongs to. The IOCTL requires the logical address of the related DoIP entity as input parameter. This logical address shall be unique within a DoIP module.

The function returns after the response message has been received or when an error has occurred.

InputData: LogicalAddress of the DoIP entity for reading the status.

ItemType: PDU\_IT\_IO\_ENTITY\_ADDRESS

pData: PDU\_IO\_ENTITY\_ADDRESS\_DATA

OutputData: Entity status as reported by the addressed DoIP entity.

ItemType: PDU\_IT\_IO\_ENTITY\_STATUS

pData: PDU\_IO\_ENTITY\_STATUS\_DATA

#### 8.5.22.2 PDU\_IOCTL\_GET\_ENTITY\_STATUS return values

Table 64 specifies specific return values for PDU\_IOCTL\_GET\_ENTITY\_STATUS.

**Table 64 — PDU\_IOCTL\_GET\_ENTITY\_STATUS return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_DOIP_AMBIGUOUS_LOGICAL_ADDRESS	Invalid or ambiguous logical address specified.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_DOIP_RESPONSE_TIMEOUT	DoIPCtrlTimeout has occurred.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.5.23 PDU\_IOCTL\_GET\_DIAGNOSTIC\_POWER\_MODE

### 8.5.23.1 General

For reading the entity status of a DoIP entity. This IOCTL should be executed on the DoIP module the specific DoIP entity belongs to. The IOCTL requires the logical address of the related DoIP entity as input parameter. This logical address shall be unique within a DoIP module.

The function returns after the response message has been received or when an error has occurred.

InputData: Logical Address of a DoIP entity for reading the power mode.  
 ItemType: PDU\_IT\_IO\_ENTITY\_ADDRESS  
 pData: PDU\_IO\_ENTITY\_ADDRESS\_DATA  
 OutputData: Recent diagnostic power mode as reported by the addressed DoIP entity.  
 ItemType: PDU\_IT\_IO\_UNUM32  
 pData: UNUM32 PowerMode; /\* Power mode of DoIP entity: 0x0: not ready  
 0x1: ready  
 0x2: not supported \*/

### 8.5.23.2 PDU\_IOCTL\_GET\_DIAGNOSTIC\_POWER\_MODE return values

Table 65 specifies specific return values for PDU\_IOCTL\_GET\_DIAGNOSTIC\_POWER\_MODE.

**Table 65 — PDU\_IOCTL\_GET\_DIAGNOSTIC\_POWER\_MODE return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_DOIP_AMBIGUOUS_LOGICAL_ADDRESS	Invalid or ambiguous logical address specified.
PDU_ERR_INVALID_PARAMETERS	At least one of the parameters is invalid (pInputData and/or pOutputData).
PDU_ERR_DOIP_RESPONSE_TIMEOUT	DoIPCtrlTimeout has occurred.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

## 8.5.24 PDU\_IOCTL\_GET\_ETH\_PIN\_OPTION

### 8.5.24.1 General

ISO 13400-3 defines different Ethernet pin layout options for the OBDII connector, e.g. Option 1 and Option 2. The IOCTL command PDU\_IOCTL\_GET\_ETH\_PIN\_OPTION is used to determine the Ethernet option. (For this determination, the Ethernet activation pin on the DLC connector is used.) Applications are encouraged to determine the Ethernet option prior to any connection to the Ethernet pins. Incorrect pin selection can cause the MVCI module to connect to non-Ethernet physical layer and cause damage to the MVCI module or the vehicle transceiver.

InputData: Pin number of the vehicles data link connector which corresponds to the vehicle Ethernet activation line.  
 ItemType: PDU\_IT\_IO\_UNUM32

pData: UNUM 32 /\*DLCPinNumber \*/  
 OutputData: Evaluated option of Ethernet pinout on the vehicle.  
 ItemType: PDU\_IT\_IO\_UNUM32  
 pData: UNUM32 EthernetPinoutOption; /\* Evaluated option of Ethernet pinout on the vehicle.  
 0 = non-Ethernet vehicle  
 1 = Ethernet Option 1  
 2 = Ethernet Option 2 \*/  
 3 and above reserved for future use

**8.5.24.2 PDU\_IOCTL\_GET\_ETH\_PIN\_OPTION return values**

Table 65 specifies specific return values for PDU\_IOCTL\_GET\_ETH\_PIN\_OPTION.

**Table 66 — PDU\_IOCTL\_GET\_ETH\_PIN\_OPTION return values**

Definition	Description
PDU_ERR_PDUAPI_NOT_CONSTRUCTED	D-PDU API has not been constructed before.
PDU_ERR_COMM_PC_TO_VCI_FAILED	Communication between host and MVCI protocol module failed.
PDU_ERR_INVALID_HANDLE	Invalid MVCI protocol module Handle or ComLogicalLink handle.
PDU_ERR_MODULE_NOT_CONNECTED	MVCI protocol module has not been connected. See PDUModuleConnect function.
PDU_ERR_FCT_FAILED	Function call failed.
PDU_STATUS_NOERROR	Function call successful.

**8.6 API functions — Error handling**

**8.6.1 Synchronous error handling**

Errors which occur during the execution of a function (i.e. synchronously) will be reported by the function's return value. The specific return values are documented for each API function in 8.4.

In D.3, reserved return values of the API functions are documented. These reserved values are supposed to provide a standard mechanism for handling errors between different D-PDU API implementations.

**8.6.2 Asynchronous error handling**

Asynchronous errors are reported by event items (i.e. items PDU\_EVENT\_ITEM with type PDU\_IT\_ERROR). They are queued by the D-PDU API using the regular internal queuing mechanisms. The application will get error items using the same functions as for result items (i.e. PDUGetEventItem). Asynchronous errors can be related to a MVCI protocol module (e.g. hardware errors), to a ComLogicalLink (e.g. CAN bus error) or to a specific ComPrimitive (e.g. ECU timeout). This relationship will be expressed by the specific handle, which is used in the function PDUGetEventItem to get the error item. The error item contains a D-PDU API defined error code (see D.4), which identifies the error that occurred along with a vendor-specific extra error code. A text translation of the vendor-specific extra error codes is available in the MDF XML file.

In D.4, reserved error codes for error items are documented. These reserved codes provide a standard mechanism for handling most asynchronous error events between different D-PDU API implementations.

**8.7 Installation**

**8.7.1 Generic description**

The D-PDU API root description file (RDF) is the central entry point for all applications accessing MVCI protocol modules in either Windows or Linux. See F.1 for the UML description of the root description file.

During installation, an MVCI protocol module Tool Integrator (see 8.3.2) adds the vendor-specific information to the root description file. If the root file has not been previously created, then the installation process shall create the root description file (and the registry information for Windows) in the specified location (see subclauses below).

The MVCI protocol module vendor shall ensure that the information to be added to the root description file does not conflict with another entry. The Tool Integrator will have to ensure uniqueness. The <MVCI\_PDU\_API> element contains the following sub-elements to help ensure unique entries:

- SHORT\_NAME;
- DESCRIPTION;
- SUPPLIER\_NAME.

The following subclauses describe the location of the root description file and the loading process of the associated libraries.

## 8.7.2 Windows installation process

### 8.7.2.1 Locating the Windows D-PDU API DLL

The application will be able to locate all of the D-PDU API implementations installed on the machine by accessing the D-PDU API root file. The location of the root file is to be identified as follows.

- a) The application shall navigate to the registry key HKEY\_LOCAL\_MACHINE\SOFTWARE\D-PDU API.
- b) Under this key, the value-name "Root File" (String) contains the full path to the root file.

EXAMPLE [HKEY\_LOCAL\_MACHINE\SOFTWARE\D-PDU API]  
"Root File"="C:\Program Files\D-PDU API\pdu\_api\_root.xml"

NOTE The complete file path to the D-PDU API Root File is stored at the defined location in the registry.

- c) Only one Key (D-PDU API) and one Value (Root File) shall be created. The pdu\_api\_root.xml file contains all the installed MVCI protocol modules' DLL information from each vendor. The uninstall program shall remove its information from the pdu\_api\_root.xml file, but shall not affect the other entries.

### 8.7.2.2 Loading the Windows D-PDU API DLL

To load the D-PDU API DLL, the application will use native Win32 API functions such as

- LoadLibrary,
- GetProcAddress:

When using GetProcAddress, the application shall supply the name of the function whose address is being requested. To support this method with un-mangled names (when using certain compilers), the MVCI protocol module vendor shall compile the DLL with an export library definition file.

- FreeLibrary.

NOTE 1 See the Win32 API SDK reference for the details of these functions.

NOTE 2 All D-PDU API functions exported from the DLL will have the \_\_stdcall calling convention.

### 8.7.3 Linux installation process

#### 8.7.3.1 Locating the Linux D-PDU API shared library

The application may locate all the D-PDU API implementations installed on the machine by accessing the D-PDU API root file. The location of the root file is to be identified as follows.

- a) The root file is stored as the file “pdu\_api\_root.xml” in the directory “/etc”.
- b) The pdu\_api\_root.xml file contains all the installed MVCI protocol modules’ shared library information from each vendor. The uninstall program shall remove its information from the pdu\_api\_root.xml file, but shall not affect the other entries.

#### 8.7.3.2 Loading the Linux D-PDU API shared library

To load the D-PDU API shared library, the application will use functions like

- dlopen,
- dlsym:

When using the function dlsym, the application shall supply the name of the function whose address is being requested, and

- dlclose.

NOTE See the Linux documentation for the details of these functions.

### 8.7.4 Selecting MVCI protocol modules

The client application should use the pdu\_api\_root.xml file to determine the list of available D-PDU API implementations. Once the application has selected one or more implementations, the pdu\_api\_root.xml file is used to retrieve all the information regarding the implementation so that the appropriate DLLs or shared libraries can be loaded for use.

## 8.8 Application notes

### 8.8.1 Interaction with the MDF

Both the D-PDU API and the application may read the MDF file to retrieve information.

### 8.8.2 Accessing additional hardware features for MVCI protocol modules

Additional hardware features (e.g. analogue channels, digital I/O, etc.), which are not covered directly by the standard D-PDU API functions, can be also implemented using standard D-PDU API function calls. The following points shall serve as a guideline for manufacturer-specific implementations of additional hardware features. However, the manufacturer is free to choose between several implementation approaches, which are outlined in the following sentences.

- The manufacturer defines a specific “HARDWARE” protocol in the MDF. This is defined the same way as for diagnostic protocols like ISO 15765.
- The “HARDWARE” protocol has a set of ComParams. These are also described in the MDF.
- To use the additional hardware features, a ComLogicalLink with the specific “HARDWARE” protocol is created by the application. Thereafter, the hardware features can be accessed either by getting and setting ComParams or by starting ComPrimitives.
- To control simple hardware features (like setting/reading digital I/O), the ComParam method (PDUSetComParam) might be sufficient. In this case, the ComParam value will carry the specific information (e.g. value of digital I/O port).

- To control advanced hardware features, the method using ComPrimitives might be a good approach. In this case, all standard features of ComPrimitives like periodic send or receive operation can be used (e.g. periodic reading analogue values). Also, more information can be exchanged between the MVCI protocol module and the application using the ComPrimitive data and its result data items.

### 8.8.3 Documentation and information provided by MVCI protocol module vendors

Each MVCI protocol module vendor will provide a different name implementation of the files supplied with the installation: module description file(s), D-PDU API library and cable description file(s). Since a number of D-PDU API implementations could simultaneously reside on the same PC, a MVCI protocol module vendor shall not name any of its files "PDU\_API.dll" nor "PDU\_API.so". The following rules shall be followed for naming each of the files delivered.

- MDF\_<VendorName>[<XXX>].xml
- CDF\_<VendorName>[<XXX>].xml
- PDUAPI\_<VendorName>[<XXX>].dll
- PDUAPI\_<VendorName>[<XXX>].so

where

- <VendorName> is the name of the vendor;
- <XXX> is an optional string (vendor specific).

#### EXAMPLES

- MDF\_DoctorWho\_V\_1\_0\_1.xml;
- CDF\_Automan\_1\_0.xml;
- PDUAPI\_Bob\_Ver\_2\_1\_0.dll.

The protocol documentation listed below will be provided.

- a) A tool manufacturer shall document the protocol behaviour and the ComParams for each protocol supported by the tool manufacturer.
- b) The documentation shall describe the behaviour of the protocol with regard to the specified ComPrimitive types and status values.
- c) All protocol ComParams shall be documented.
- d) The tool manufacturer shall provide the protocol-specific entries in the MDF.

The vendor will also supply the OptionStr for PDUConstruct. The string provides a list of attributes and their values, which are specific to a D-PDU API implementation (see 8.4.2.4 for more information).

## 9 Using the D-PDU API with existing applications

### 9.1 SAE J2534-1 and RP1210a existing standards

The standards SAE J2534-1 and RP1210a were defined prior to the D-PDU API and applications have already been introduced to the aftersales market. In order to preserve the applications based on SAE J2534-1 and RP1210a, an MVCI-compliant device (i.e. MVCI protocol module) can be made convertible to SAE J2534-1 or RP1210a with a compatibility layer or wrapper. This library configuration will enable SAE J2534-1 and RP1210a applications to run on a MVCI compliant device. See Figure 28.

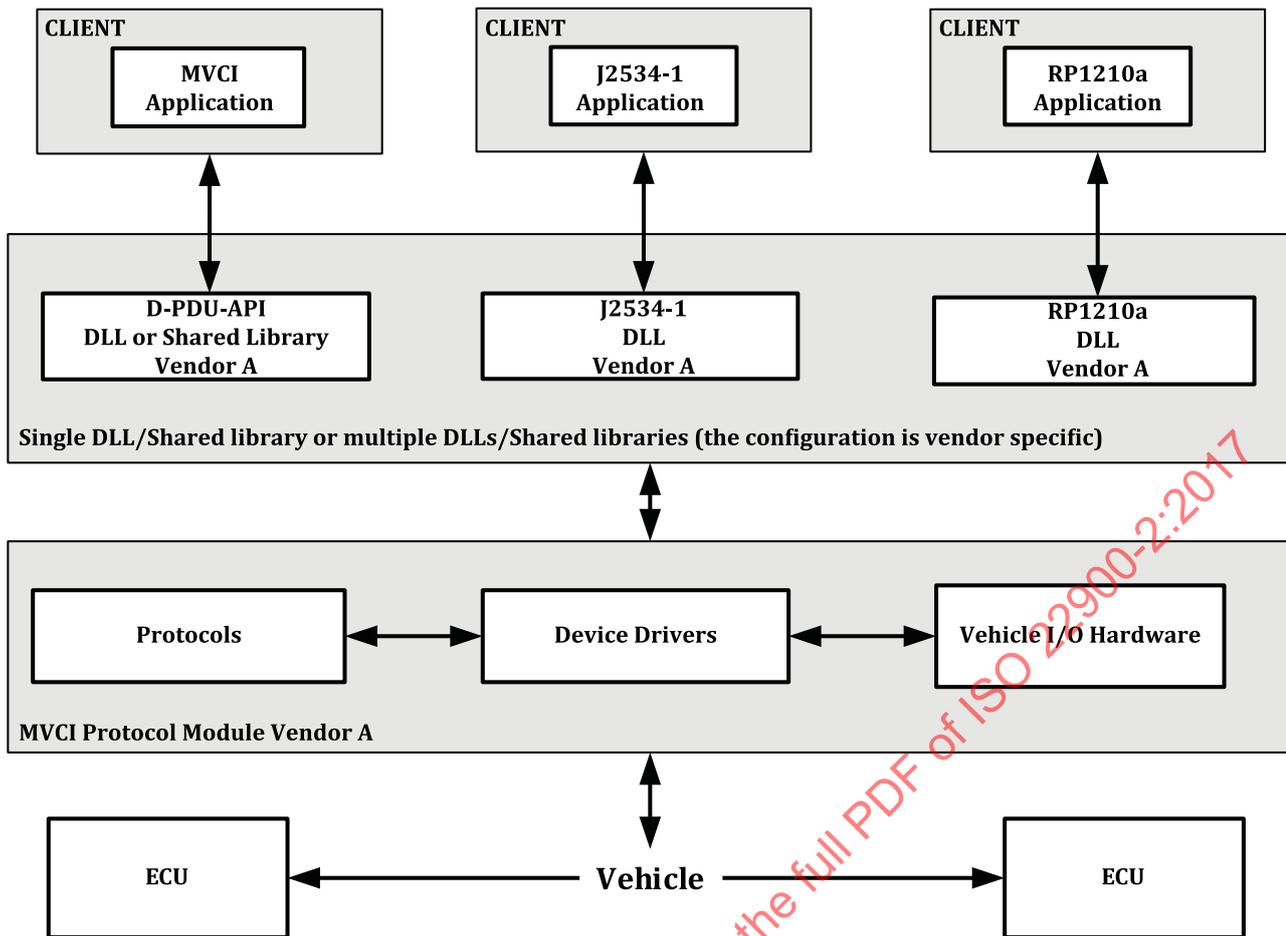


Figure 28 — Adapting MSCI device to SAE J2534-1 and RP1210a API

## 10 Data structures

### 10.1 API functions — Data structure definitions

#### 10.1.1 Abstract basic data types

For all input/output parameters, the following subset of abstract basic data types shall be used (see Table 67).

Table 67 — Abstract data types

Data type	Description
UNUM8	Unsigned numeric 8 bits.
SNUM8	Signed numeric 8 bits.
UNUM16	Unsigned numeric 16 bits.
SNUM16	Signed numeric 16 bits.
UNUM32	Unsigned numeric 32 bits.
SNUM32	Signed numeric 32 bits.

In addition to these data types, the following abstract data types shall be used for string handling (see Table 68).

**Table 68 — Abstract data types — string handling**

Data type	Description
CHAR8	ASCII-coded 8-bit character value (ISO 8859-1 (Latin 1)).

All strings shall be handled as zero-terminated character field of the appropriate character data type. Length information is calculated without the zero termination character value.

### 10.1.2 Definitions

Table 69 specifies the following definitions used for D-PDU API functions.

**Table 69 — Definitions for D-PDU API functions**

Definition	Description
EXTERNC	Extern "C" declaration, required for C++ code.
CALLBACKFNC	Callback function type.

These definitions shall be defined according to the requirements of the specific C/C++ compiler.

In a Windows D-PDU API DLL, all D-PDU API functions will have the `stdcall` calling convention in accordance with 9.7.2.

### 10.1.3 Bit encoding for UNUM32

Table 70 describes how to set or read a parameter field which contains a bit encoded field. This table has been added to help with Endian problems between different hardware platforms.

**Table 70 — Definition of byte and bit position for UNUM32**

BYTE 3 (MSB)								BYTE 2								BYTE 1								BYTE 0 (LSB)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

### 10.1.4 API data structures

#### 10.1.4.1 General

The following data structures are required for data transfer of D-PDU API functions. Byte packing of one byte is adopted from SAE J2534-1. This has to be defined according to the requirements of the specific C/C++ compiler.

#### 10.1.4.2 Generic item for type evaluation and casting

This is a generic item used for casting to item-specific structures. `PDU_ITEM` is used in the function `PDU_DestroyItem`.

```
typedef struct {
    T_PDU_IT    ItemType;    /* see D.1.1 */
} PDU_ITEM;
```

Table 71 lists the different type of Item types and their respective structures.

Table 71 — Item type structures for typecasting

Item types	Item type specific structure
PDU_IT_IO_UNUM32	PDU_DATA_ITEM, void *pData contains a single UNUM32 value.
PDU_IT_IO_PROG_VOLTAGE	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_PROG_VOLTAGE_DATA structure.
PDU_IT_IO_BYTEARRAY	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_BYTEARRAY_DATA structure.
PDU_IT_IO_FILTER	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_FILTER_LIST structure.
PDU_IT_IO_EVENT_QUEUE_PROPERTY	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_EVENT_QUEUE_PROPERTY_DATA structure.
PDU_IT_IO_VEHICLE_ID_REQUEST	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_VEHICLE_ID_REQUEST structure.
PDU_IT_IO_ETH_SWITCH_STATE	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_ETH_SWITCH_STATE structure.
PDU_IT_IO_ENTITY_ADDRESS	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_ENTITY_ADDRESS_DATA structure.
PDU_IT_IO_ENTITY_STATUS	PDU_DATA_ITEM, void *pData contains a pointer to the PDU_IO_ENTITY_STATUS_DATA structure.
PDU_IT_RSC_STATUS	PDU_RSC_STATUS_ITEM
PDU_IT_PARAM	PDU_PARAM_ITEM
PDU_IT_RESULT	PDU_RESULT_DATA
PDU_IT_STATUS	PDU_STATUS_DATA
PDU_IT_INFO	PDU_INFO_DATA
PDU_IT_ERROR	PDU_ERROR_DATA
PDU_IT_RSC_ID	PDU_RSC_ID_ITEM
PDU_IT_RSC_CONFLICT	PDU_RSC_CONFLICT_ITEM
PDU_IT_MODULE_ID	PDU_MODULE_ITEM
PDU_IT_UNIQUE_RESP_ID_TABLE	PDU_UNIQUE_RESP_ID_TABLE_ITEM

### 10.1.4.3 Items for IOCTL data transfer (PDUIoctl)

#### 10.1.4.3.1 Item for Generic IOCTL data item

This is a generic IOCTL data item used for casting to specific IOCTL type structures. PDU\_DATA\_ITEM is used in the function PDUIoctl.

```
typedef struct {
    T_PDU_IT ItemType;           /* value= one of the IOCTL constants from D.1.1 */
    void *pData;                /* pointer to the specific IOCTL data structure */
} PDU_DATA_ITEM;
```

#### 10.1.4.3.2 IOCTL programming voltage structure (PDU\_IT\_IO\_PROG\_VOLTAGE)

```
typedef struct {
    UNUM32 ProgVoltage_mv;      /* programming voltage in mV */
    UNUM32 PinOnDLC;           /* pin number on Data Link Connector */
} PDU_IO_PROG_VOLTAGE_DATA;
```

**10.1.4.3.3 IOCTL byte array structure (PDU\_IT\_IO\_BYTEARRAY)**

```
typedef struct {
    UNUM32    DataSize;          /* number of bytes in the data array */
    UNUM8     *pData;           /* pointer to the data array */
} PDU_IO_BYTEARRAY_DATA;
```

**10.1.4.3.4 IOCTL filter list structure (PDU\_IT\_IO\_FILTER)**

```
typedef struct {
    UNUM32 NumFilterEntries;     /* number of Filter entries in the filter list array */
    PDU_IO_FILTER_DATA *pFilterData; /* pointer to an array of filter data */
} PDU_IO_FILTER_LIST;
```

**10.1.4.3.5 IOCTL filter data structure**

```
typedef struct {
    T_PDU_FILTER    FilterType;     /* type of filter being configured. D.1.10 */
    UNUM32 FilterNumber;           /* Filter Number. Used to replace filters and stop filters. Range depends on
    implementation (see 9.5.13 PDU_IOCTL_START_MSG_FILTER) */
    UNUM32 FilterCompareSize;     /* Number of bytes used out of each of the filter messages arrays Range 1-12. */
    UNUM8 FilterMaskMessage[12];  /* Mask message to be ANDed to each incoming message. When using the CAN
    protocol, setting the first 4 bytes of FilterMaskMessage to 0xFF makes the filter
    specific to one CAN ID. Using other values allows for the reception or blocking of
    multiple CAN identifiers. */
    UNUM8 FilterPatternMessage[12]; /* Pattern message to be compared to the incoming message after the
    FilterMaskMessage has been applied. If the result matches this pattern message
    and the FilterType is a pass filter, then the incoming message will be processed
    for further reception (otherwise it will be discarded). If the result matches this
    pattern message and the FilterType is a block filter, then the incoming message
    will be discarded (otherwise it will be processed for further reception). Message
    bytes in the received message that are beyond the FilterCompareSize of the
    pattern message will be treated as "don't care". */
} PDU_IO_FILTER_DATA;
```

**10.1.4.3.6 IOCTL event queue property structure (PDU\_IT\_IO\_EVENT\_QUEUE\_PROPERTY)**

```
typedef struct {
    UNUM32 QueueSize;           /* maximum size of event queue */
    T_PDU_QUEUE_MODE QueueMode; /* Queue mode. D.1.11 */
} PDU_IO_EVENT_QUEUE_PROPERTY_DATA;
```

**10.1.4.3.7 IOCTL vehicle ID request structure (PDU\_IOCTL\_VEHICLE\_ID\_REQUEST)**

```
typedef struct {
    UNUM32 PreselectionMode; /* Preselection mode: 0=no preselection */
    /* 1= select DoIP entities with given VIN */
    /* 2= select DoIP entities with given EID */
    CHAR8* PreselectionValue; /* pointer to NULL terminated ASCII string */
    /* containing optional VIN or EID (depending on */

```

```

        /* PreselectionMode) */
UNUM32  CombinationMode; /* Combination mode: */
        /* 0= no combination */
        /* 1= combine DoIP entities with common VIN */
        /* into MVCI of type "DoIP-Vehicle" */
        /* 2= combine DoIP entities with common GroupID */
        /* into MVCI of type "DoIP-Group" */
        /* 3= combine all DoIP entities */
        /* into MVCI of type "DoIP-Collection" */
UNUM32  VehicleDiscoveryTime; /* Time-out to wait for vehicle identification responses. 0=return immediately, or
time in milliseconds. */
UNUM32  NumDestinationAddresses /* Number of broadcast/multicast addresses in the destination address array.
May be 0 for default */
PDU_IP_ADDR_INFO *pDestinationAddresses /* pointer to an array of IP addresses on which broadcast/multicast
should be performed */
} PDU_IO_VEHICLE_ID_REQUEST;

typedef struct {
    UNUM32  IpVersion; /* the IP version to be used for DoIP communication: 4=IPv4, 6=IPv6 */
    UNUM8   *pAddress /* IPv4: 4 Byte broadcast address (in network byte order; MSB first)
                    IPv6: 16 Byte multicast address (in network byte order; MSB first)
                    */
} PDU_IP_ADDR_INFO;

```

#### 10.1.4.3.8 IOCTL set Ethernet switch state structure (PDU\_IOCTL\_SET\_ETH\_SWITCH\_STATE)

```

typedef struct {
    UNUM32  EthernetSenseState; /*EthernetSenseState
                                0 = Ethernet activation pin off
                                1 = Ethernet activation pin on*/
    UNUM32  EthernetActPinNumber /* EthernetActPinNumber
                                Pin number on DLC of the Ethernet activation pin. Default shall be 8 as defined in
                                ISO 13400-3 for OBD-connector */
} PDU_IO_ETH_SWITCH_STATE;

```

#### 10.1.4.4 Item for resource status information (used by PDUGetResourceStatus)

```

typedef struct {
    T_PDU_IT  ItemType; /* value= PDU_IT_RSC_STATUS (IN parameter)*/
    UNUM32  NumEntries; /* (IN Parameter) = number of entries in
                        pResourceStatusData array. */
    PDU_RSC_STATUS_DATA *pResourceStatusData; /* array to contain resource status (IN Parameter)*/
} PDU_RSC_STATUS_ITEM;

```

```

typedef struct {
    UNUM32  hMod; /* Handle of a MVCI protocol module (IN parameter) */
    UNUM32  ResourceId /* Resource ID (IN parameter) */
    UNUM32  ResourceStatus; /* Resource Information Status (OUT Parameter): (see D.1.6 for specific
                            values.)*/
} PDU_RSC_STATUS_DATA;

```

**10.1.4.5 Item for ComParam data (used by PDUGetComParam, PDUSetComParam)**

```
typedef struct {
    T_PDU_IT  ItemType;          /* value= PDU_IT_PARAM */
    UNUM32   ComParamId;        /* ComParam Id. Value from MDF of MVCI protocol module */
    T_PDU_PT  ComParamDataType; /* Defines the data type of the ComParam B.3.3 */
    T_PDU_PC  ComParamClass;    /*ComParam Class type. The class type is used by the D-PDU API for
                                special ComParam handling cases. (BusType (physical ComParams) and
                                Unique ID ComParams)). See B.3.2 */
    Void      *pComParamData;   /* pointer to ComParam data of type ComParamDataType */
} PDU_PARAM_ITEM;
```

**10.1.4.6 Item for module identification (used by PDUGetModuleIds)**

```
typedef struct {
    T_PDU_IT  ItemType;          /* value= PDU_IT_MODULE_ID */
    UNUM32    NumEntries;        /* number of entries written to the pModuleData array */
    PDU_MODULE_DATA *pModuleData; /* pointer to array containing module types and module handles */
} PDU_MODULE_ITEM;
```

```
typedef struct {
    UNUM32    ModuleTypeId;      /* MVCI protocol moduleTypeId */
    UNUM32    hMod;              /* handle of MVCI protocol module assigned by D-PDU API */
    CHAR8     *pVendorModuleName; /* Vendor specific information string for the unique module identification,
                                e.g. Module serial number or user friendly name */
    CHAR8     *pVendorAdditionalInfo; /* Vendor specific additional information string */
    T_PDU_STATUSModuleStatus;    /* Status of MVCI protocol module detected by D-PDU API session.
                                D.1.4 */
} PDU_MODULE_DATA;
```

**pVendorModuleName and pVendorAdditionalInfo string description:**

The pVendorModuleName and pVendorAdditionalInfo strings contain a list of attributes and their values. An attribute and its corresponding value are to be separated by a >=< sign. The value needs to be put inside two >'< signs. Between pairs of attribute and value shall be at least one space character. Attributes and values are specific to a D-PDU API vendor implementation.

When no string information is available, the pVendorxxx strings will be set to NULL.

EXAMPLE String

```
pVendorModuleName = "VendorName='MVCI Company' MVCI Friendly Name = 'Hugo' "
pVendorAdditionalInfo = "Connection Type Wireless='xx.xx.xx.xx' "
```

The additional information can contain the IP addresses and connection types available:

- a) Ethernet;
- b) Bluetooth;
- c) Infrared;
- d) 802.11g/802.11b;
- e) 802.3;

- f) RS232;
- g) USB;
- h) PCMCIA;
- i) PCI Express;
- j) WAN (GPRS, UMTS, ...).

#### 10.1.4.7 Items for resource identification (used by PDUGetResourceIds)

```
typedef struct {
    T_PDU_IT ItemType; /* value = PDU_IT_RSC_ID (IN parameter)*/
    UNUM32 NumModules; /* number of entries in pResourceIdDataArray. */
    PDU_RSC_ID_ITEM_DATA *pResourceIdDataArray; /* pointer to an array of resource Id Item Data */
} PDU_RSC_ID_ITEM;
```

```
typedef struct {
    UNUM32 hMod; /* MSCI protocol module Handle */
    UNUM32 NumIds; /* number of resources that match PDU_RSC_DATA */
    UNUM32 *pResourceIdArray; /* pointer to a list of resource ids */
} PDU_RSC_ID_ITEM_DATA;
```

#### 10.1.4.8 Structure for resource data (used by PDUGetResourceIds and PDUCreateComLogicalLink)

```
typedef struct {
    UNUM32 BusTypeId; /* Bus Type Id (IN parameter) */
    UNUM32 ProtocolId; /* Protocol Id (IN parameter) */
    UNUM32 NumPinData; /* Number of items in the following array */
    PDU_PIN_DATA *pDLCPinData; /* Pointer to array of PDU_PIN_DATA structures */
} PDU_RSC_DATA;
```

PDU\_PIN\_DATA is defined as:

```
typedef struct {
    UNUM32 DLCPinNumber; /* Pin number on DLC */
    UNUM32 DLCPinTypeId; /* Pin ID */
} PDU_PIN_DATA;
```

#### 10.1.4.9 Item for conflicting resources (used by PDUGetConflictingResources)

```
typedef struct {
    T_PDU_IT ItemType; /* value= PDU_IT_RSC_CONFLICT */
    UNUM32 NumEntries; /* number of entries written to pRscConflictData*/
    PDU_RSC_CONFLICT_DATA *pRscConflictData; /* pointer to array of PDU_RSC_CONFLICT_DATA*/
} PDU_RSC_CONFLICT_ITEM;
```

```
typedef struct {
    UNUM32 hMod; /* Handle of the MSCI protocol module with conflict*/
    UNUM32 ResourceId; /* Conflicting Resource ID */
} PDU_RSC_CONFLICT_DATA;
```

#### 10.1.4.10 Item for unique response identification (used by PDUGetUniqueRespIdTable and PDUSetUniqueRespIdTable)

```
typedef struct {
    T_PDU_IT ItemType;          /* value= PDU_IT_UNIQUE_RESP_ID_TABLE */
    UNUM32 NumEntries; /* number of entries in the table */
    PDU_ECU_UNIQUE_RESP_DATA *pUniqueData; /* pointer to array of table entries for each ECU response */
} PDU_UNIQUE_RESP_ID_TABLE_ITEM;

typedef struct {
    UNUM32 UniqueRespIdentifier; /* filled out by application */
    UNUM32 NumParamItems; /* number of ComParams for the Unique Identifier */
    PDU_PARAM_ITEM *pParams; /* pointer to array of ComParam items to uniquely define a ECU response.
                               The list is protocol specific */
} PDU_ECU_UNIQUE_RESP_DATA;
```

EXAMPLE Table 72 and Table 73 are examples for URID table configurations.

**Table 72 — Example set of ComParams for an ISO 15765 protocol response address ComParam list**

ComParam	Description
CP_CanPhysReqFormat	CAN Id format for a physical request. Used for Flow Control Can transmission as well. First entry in the Unique Response Identifier table is the default entry for a physical request.
CP_CanPhysReqId	CAN Id for physical request. Used for Flow Control Can transmission as well. First entry in the Unique Response Identifier table is the default entry for a physical request.
CP_CanPhysReqExtAddr	Can extended address for physical request. Used for Flow Control Can transmission as well. First entry in the Unique Response Identifier table is the default entry for a physical request.
CP_CanRespUSDTFormat	CAN Id format for a USDT response. Used for response handling.
CP_CanRespUSDTId	CAN Id for a USDT response. Used for response handling. Value set to 0xFFFFFFFF is not used.
CP_CanRespUSDTExtAddr	CAN extended address for a USDT response. Used for response handling.
CP_CanRespUUDTFormat	CAN Id format for a UUDT response. Used for response handling.
CP_CanRespUUDTExtAddr	CAN Id extended address for a UUDT response. Used for response handling.
CP_CanRespUUDTId	CAN Id for a UUDT response. Used for response handling. Value set to 0xFFFFFFFF is not used.

**Table 73 — Example set of ComParams for a SAE J2190 (non-CAN) response structure per ECU**

ComParam	Description
CP_FuncRespFormatPriorityType	First byte of received message is the format/priority byte. This is the expected first byte on a functional response.
CP_FuncRespTargetAddr	Second byte of received message for a functional addressed response. The protocol handler will use either the Tester Source address or this FuncRespTargetAddr (e.g. OBD using 0x6B as the Target address instead of the Tester Address).
CP_PhysRespFormatPriorityType	First byte of received message is the format/priority byte. This is the expected first byte on a physical response.
CP_EcuRespSourceAddress	Expected ECU Source Address. This is typically the third byte of the message.

**10.1.4.11 Item for event notification**

```
typedef struct {
    T_PDU_IT    ItemType;          /* value= PDU_IT_RESULT or PDU_IT_STATUS or PDU_IT_ERROR or
                                   PDU_IT_INFO */
    UNUM32    hCop;                /* If item is from a ComPrimitive then the hCop contains the valid ComPrimitive handle,
                                   else it contains PDU_HANDLE_UNDEF */
    void *    pCoPTag;            /* ComPrimitive Tag. Should be ignored if hCop = PDU_HANDLE_UNDEF */
    UNUM32    Timestamp;          /* Timestamp in microseconds */
    void *pData;                  /* points to the data for the specified Item Type. See 10.1.4.11.1 to 10.1.4.11.4 */
} PDU_EVENT_ITEM;
```

A successful call to PDUGetEventItem will automatically remove the item from the top of the event queue in the D-PDU API. The application shall call PDUDestroyItem to release the memory back to the D-PDU API.

**10.1.4.11.1 Structure for status data**

Asynchronous status change notification for PDU\_IT\_STATUS Item.

```
T_PDU_STATUS PDU_STATUS_DATA;    /* Status code information. D.1.4 */
```

**10.1.4.11.2 Asynchronous event information notification for PDU\_IT\_INFO Item**

```
typedef struct {
    T_PDU_INFO    InfoCode;        /* Information code. D.1.5 */
    UNUM32    ExtraInfoData;      /* Optional additional information */
} PDU_INFO_DATA;
```

**10.1.4.11.3 Asynchronous error notification structure for the PDU\_IT\_ERROR Item**

```
typedef struct {
    T_PDU_ERR_EVT    ErrorCodeId;  /* error code, binary information. */
    UNUM32    ExtraErrorInfoId;    /* Optional additional error information, text translation via MDF file. Binary
                                   Information, 0 indicates no additional error information. See D.4 */
} PDU_ERROR_DATA;
```

**10.1.4.11.4 Structure for result data**

Asynchronous result notification structure (received data) for the PDU\_IT\_RESULT Item.

```
typedef struct {
    PDU_FLAG_DATA    RxFlag;        /* Receive message status. See D.2.2.*/
    UNUM32    UniqueRespIdentifier; /* ECU response unique identifier */
    UNUM32    AcceptanceId;        /* Acceptance Id value from ComPrimitive Expected Response Structure. If
                                   multiple expected response entries match the response payload data, then
                                   the first matching expected response id found in the array of expected
                                   responses is used (i.e. acceptance filtering is carried out in the sequence of
                                   the expected responses as they appear in the array of expected responses.
                                   Thus, an expected response with the lowest array index has the highest
                                   priority.)*/
    PDU_FLAG_DATA    TimestampFlags; /* Bitoriented Timestamp Indicator flag (see 10.1.4.13 and
                                   D.2.4). If the flag data is 0, then the following timestamp
```

```

information is not valid.*/
UNUM32 TxMsgDoneTimestamp; /* Transmit Message done Timestamp in microseconds */
UNUM32 StartMsgTimestamp; /* Start Message Timestamp in microseconds */
PDU_EXTRA_INFO *pExtraInfo; /* If NULL, no extra information is attached to the response structure. This
feature is enabled by setting the ENABLE_EXTRA_INFO bit in the TxFlag for
the ComPrimitive (see D.2.1)*/

UNUM32 NumDataBytes; /* Data size in bytes, if RawMode then the data includes header bytes,
checksum, message data bytes (pDataBytes), and extra data, if any.*/

UNUM8 *pDataBytes; /* Reference pointer to D-PDU API memory that contains PDU Payload data.
In non-Raw mode this data contains no header bytes, CAN Ids, or checksum
information. In RawMode, this data will contain the exact data received from
the ECU. For ISO 15765, ISO 11898 and SAE J1939, the first 4 bytes are
the CAN ID (11 bit or 29 bit) followed by a possible extended address byte
(Table D.4) */

} PDU_RESULT_DATA;

```

#### 10.1.4.12 Structure for extra result data information

```

typedef struct {
    UNUM32 NumHeaderBytes; /* Number of header bytes contained in pHeaderBytes array.*/
    UNUM32 NumFooterBytes; /* Number of footer bytes contained in pFooterBytes array. (SAE J1850
PWM) Start position of extra data in received message (for example, IFR or
ISO 14230 checksum.) When no extra data bytes are present in the
message, NumFooterBytes shall be set to zero.*/

    UNUM8 *pHeaderBytes; /* Reference pointer to Response PDU Header bytes, NULL if
NumHeaderBytes = 0 */

    UNUM8 *pFooterBytes; /* Reference pointer to Response PDU Footer bytes, NULL if
NumFooterBytes = 0 */
} PDU_EXTRA_INFO;

```

#### 10.1.4.13 Structure for flag data

```

typedef struct {
    UNUM32 NumFlagBytes; /* number of bytes in pFlagData array*/
    UNUM8 *pFlagData; /* Pointer to flag bytes used for TxFlag, RxFlag, and CllCreateFlag. See D.2 */
} PDU_FLAG_DATA;

```

#### 10.1.4.14 Structure for version information (used by PDUGetVersion)

```

typedef struct {
    UNUM32 MVCI_Part1StandardVersion; /* Release version of supported MVCI Part 1 standard
(see Coding of version numbers)*/
    UNUM32 MVCI_Part2StandardVersion; /* Release version of supported MVCI Part 2 standard
(see Coding of version numbers)*/
    UNUM32 HwSerialNumber; /* Unique Serial number of MVCI HW module from a vendor */
    CHAR8 HwName[64]; /* Name of MVCI HW module; zero terminated */
    UNUM32 HwVersion; /* Version number of MVCI HW module
(see Coding of version numbers)*/
    UNUM32 HwDate; /* Manufacturing date of MVCI HW module
(see Coding of dates)*/
    UNUM32 HwInterface; /* Type of MVCI HW module; zero terminated */
    CHAR8 FwName[64]; /* Name of the firmware available in the MVCI HW module */
}

```

```

UNUM32    FwVersion;                /* Version number of the firmware in the MVCI HW module
                                           (see Coding of version numbers)*/
UNUM32    FwDate;                  /* Manufacturing date of the firmware in the MVCI HW module
                                           (see Coding of dates)*/
CHAR8     VendorName[64];          /* Name of vendor; zero terminated */
CHAR8     PDUApiSwName[64];        /* Name of the D-PDU API software; zero terminated */
UNUM32    PDUApiSwVersion;         /* Version number of D-PDU API software
                                           (see Coding of version numbers)*/
UNUM32    PDUApiSwDate;           /* Manufacturing date of the D-PDU API software
                                           (see Coding of dates)*/
} PDU_VERSION_DATA;

```

**10.1.4.15 Coding of version numbers**

Version numbers from PDU\_VERSION\_DATA are coded as defined in Table 74.

**Table 74 — Coding of version numbers: UNUM32**

MSB		LSB	
Major (0..255)	Minor (0..255)	Revision (0..255)	0

**10.1.4.16 Coding of dates**

Date numbers from PDU\_VERSION\_DATA are coded as defined in Table 75.

**Table 75 — Coding of dates: UNUM32**

MSB		LSB	
Year since 1970 (0..255)	Month (1..12)	Day (1..31)	Week (1..52, 0 if not used)

**10.1.4.17 Structure to control a ComPrimitive's operation (used by PDUStartComPrimitive)**

PDU\_COP\_CTRL\_DATA is not applicable to ComPrimitives types PDU\_COPT\_UPDATEPARAM and PDU\_COPT\_RESTORE\_PARAM.

```

typedef struct {
    UNUM32    Time;                /* Cycle time in ms for cyclic send operation or delay time for
                                           PDU_COPT_DELAY ComPrimitive. If cyclic time is set to 0, then the
                                           ComPrimitive is put on the transmit queue after each completion cycle, but is
                                           at a lower priority than other ComPrimitives and Tester Present Messages. */
    SNUM32    NumSendCycles;        /* # of send cycles to be performed; -1 for infinite cyclic send operation */
    SNUM32    NumReceiveCycles;    /* # of receive cycles to be performed; -1 (IS-CYCLIC) for infinite receive
                                           operation, -2 (IS-MULTIPLE) for multiple expected responses from 1 or more
                                           ECUs */
    UNUM32    TempParamUpdate;     /* Temporary ComParam settings for the ComPrimitive: 0 = Do not use
                                           temporary ComParams for this ComPrimitive. The ComPrimitive shall attach
                                           the "Active" ComParam buffer to the ComPrimitive. This buffer shall be
                                           in effect for the ComPrimitive until it is finished. The ComParams for the
                                           ComPrimitive will not change even if the "Active" buffer is modified by a
                                           subsequent ComPrimitive type of PDU_COPT_UPDATEPARAM.
                                           1 = Use temporary ComParams for this ComPrimitive; The
                                           ComPrimitive shall attach the ComParam "Working" buffer to the

```

ComPrimitive. This buffer shall be in effect for the ComPrimitive until it is finished. The ComParams for the ComPrimitive will not change even if the “Active” or “Working” buffers are modified by any subsequent calls to PDUSetComParam.

NOTE 1 If TempParamUpdate is set to 1, the ComParam working buffer is restored to the active buffer when this PDUStartComPrimitive function call returns.

NOTE 2 Physical ComParams cannot be changed using the TempParamUpdate flag \*/

```
PDU_FLAG_DATA TxFlag; /* Transmit Flag used to indicate protocol specific elements for the
ComPrimitive's execution. (see D.2.1.)*/
UNUM32 NumPossibleExpectedResponses; /* number of entries in pExpectedResponseArray */
PDU_EXP_RESP_DATA *pExpectedResponseArray; /* pointer to an array of expected responses (see
see 10.1.4.19) */
```

```
} PDU_COP_CTRL_DATA;
```

#### 10.1.4.18 Data Items for IOCTL PDU\_IOCTL\_GET\_ENTITY\_STATUS

```
typedef struct {
UNUM32 LogicalAddress; /* Logical address of DoIP entity to access */
UNUM32 DoIPCtrlTimeout; /* Timeout in milliseconds to wait for the response from the DoIP entity
(corresponds to A_DoIP_Ctrl in ISO 13400) */
} PDU_IO_ENTITY_ADDRESS_DATA;

typedef struct {
UNUM32 EntityType; /* Type of DoIP entity: 0x0=gateway 0x1=node */
UNUM32 TcpClientsMax; /* Maximum number of concurrent TCP_DATA sockets allowed with this DoIP
entity, excluding the reserve socket required for socket handling. */
UNUM32 TcpClients; /* Number of currently established sockets. */
UNUM32 MaxDataSize; /* Optional limit (in bytes) for the maximum size of a single DoIP request. This
value will be 0 if no limits are reported by the DoIP entity. */
} PDU_IO_ENTITY_STATUS_DATA;
```

#### 10.1.4.19 Structure for expected response

```
typedef struct {
UNUM32 ResponseType; /* 0 = positive response; 1 = negative response */
UNUM32 AcceptanceId; /* ID assigned by application to be returned in PDU_RESULT_DATA, which
indicates which expected response matched */
UNUM32 NumMaskPatternBytes; /* number of bytes in the Mask Data and Pattern Data */
UNUM8 *pMaskData; /* Pointer to Mask Data. Bits set to a '1' are care bits, '0' are don't care bits. */
UNUM8 *pPatternData; /* Pointer to Pattern Data. Bytes to compare after the mask is applied */
UNUM32 NumUniqueRespIds; /* number of items in the following array of unique response identifiers. If the
number is set to 0, then responses with any unique response identifier are
considered, when trying to match them to this expected response. */
UNUM32 *pUniqueRespIds; /* Array containing unique response identifiers. Only responses with a unique
response identifier found in this array are considered, when trying to match
them to this expected response. */
} PDU_EXP_RESP_DATA;
```

##### 10.1.4.19.1 Expected response type

The following are the expected response types/handling.

**a) Positive Response Type (ResponseType = 0)**

The D-PDU API uses a matched positive response entry to indicate no further processing needs to be done on the complete received message (i.e. negative response handling is not checked). A complete received message from an ECU contains all frames [or protocols that use a transport/network layer (e.g. ISO 15765)] or all messages [for protocols that have CP\_EnableConcatenation on]. The received message will be sent to the client application with the associated "Acceptance Id".

NOTE 1 The positive receive message can be discarded for ISO 14229-1 if the SuppressPositiveResponse bit is set and a positive response has been sent after a negative response.

**b) Negative Response Type (ResponseType = 1)**

The D-PDU API uses a matched negative response entry to indicate that the received message may need negative response processing (see ComParams CP\_RCxxHandling). If negative response handling is not enabled for the 0x7F response code, then the message will be sent to the client application with the associated "Acceptance Id". In this case, it is the responsibility of the client application to handle the negative response message.

NOTE 2 For ISO 15765 protocols, only USDT single frames can be matched to a negative response entry, negative responses are never transmitted in UUDT frames.

**c) "Generic" negative response handling**

The D-PDU API uses a special case for negative response handling if no match is found in the expected response structure. If negative response handling is enabled (CP\_RCxxHandling is not 0), the D-PDU API follows these steps.

- 1) Does the ComLogicalLink have an active SendRecv ComPrimitive? If not discard the message.
- 2) If the protocol is of the type ISO 15765, is the CAN ID USDT? If not discard the frame.
- 3) Is the first byte of the message = 0x7F? If not discard the message.
- 4) Is the second byte of the message = to the Service Id (SID) of the active ComPrimitive? If not discard the message.
- 5) Retrieve the negative response code from the message (see CP\_RCByteOffset). Does it match one of the types enabled (see CP\_RCxxHandling)? If not, discard the message.
- 6) Process negative response message as if there were a match in the expected response structure.

**10.1.4.19.2 Array of unique response ids (pUniqueRespIds)**

The array of unique response identifiers may be used if an expected response only appears for specific unique response identifiers. This situation may occur in the case of functional addressing, where the possible responses are not common to all ECUs.

The number of unique response identifiers may be 0. In this case, the array pUniqueRespIds is not used and all responses with any unique response identifier are considered when trying to match actual response data to the expected response data.

**10.1.4.19.3 Expected response matching rules**

If multiple expected response entries match the response payload data, then the first matching expected response id found in the array of expected responses is used. (That is, acceptance filtering is carried out in the sequence of the expected responses as they appear in the array of expected responses. Thus, an expected response with the lowest array index has the highest priority.)

For acceptance filtering, the D-PDU API tries to match the data bytes of a received response to the pattern bytes of an expected response (always regarding the mask bytes). The number of data bytes in the received response may

differ from the number of mask and pattern bytes (NumMaskPatternbytes) in the expected response. Acceptance filtering uses the following rules.

- a) If the number of received data bytes is less than NumMaskPatternBytes, the response does not match the expected response.
- b) If the number of received data bytes equals NumMaskPatternBytes, all data bytes are compared with the pattern data bytes.
- c) If the number of received data bytes exceeds NumMaskPatternBytes, only the initial data bytes of the received response are compared with all pattern data bytes of the expected response. Any following data bytes in the received response are “don't care”.

This expected response structure can be used to mask for ranges of expected responses. For example, a single request to an ECU could generate a 0x7F response, a positive response, an On-Event response, a repetitive response, etc.

The NumberOfPossibleExpectedResponses could contain two entries if 0x7F responses are possible for the requested service.

#### 10.1.4.19.4 Expected response example

Example array of expected responses:

- [0] Acceptance ID: 0    MaskData: 0xFF 0xFF    PatternData: 0x5A 0x90
- [1] Acceptance ID: 1    MaskData: 0xFF    PatternData: 0x5A
- [2] Acceptance ID: 2    MaskData: 0xFF 0xFF    PatternData: 0x7F 0x1A

Example response matching:

- Received response (a): 0x5A 0x90    ---> AcceptanceId = 0
- Received response (b): 0x5A 0x90 0x31    ---> AcceptanceId = 0
- Received response (c): 0x5A    ---> AcceptanceId = 1
- Received response (d): 0x5A 0x91    ---> AcceptanceId = 1
- Received response (e): 0x5B    ---> AcceptanceId = unexpected response (discarded)
- Received response (f): 0x7F 0x1A    ---> AcceptanceId = 2
- Received response (g): 0x7F 0x5A    ---> AcceptanceId = unexpected response (discarded)

#### 10.1.4.19.5 Expected response structure (RawMode/NonRawMode)

In NonRawMode, no header bytes are returned to the application. Therefore, the expected response structure contains expected message payload data only.

In RawMode, the expected response shall include the header bytes of the expected message. For ISO 15765, ISO\_11898\_RAW and SAE J1939, the first 4 bytes will always contain the CAN ID. If extended addressing is expected from the responding ECU, then the first byte after the CAN ID contains the extended address. After any header bytes or CAN IDs, the expected message payload data can be masked for.

Table 76 describes how the PDU\_EXP\_RESP\_DATA is handled in RawMode per protocol.

Table 76 — Raw mode expected response format per protocol

Protocol	RawMode — Expected response handling description
ISO 15765	The first 4 bytes of the expected data are reserved for the CAN ID (11 bit or 29 bit). If extended addressing, then the fifth byte of the expected data contains the expected extended address.
SAE J1850_VPW, SAE J1850_PWM, ISO 9141	Expected data contain header bytes (1 to 3) followed by the payload data bytes.
ISO 14230	The number of ECU response header bytes can vary from 1 to 4 followed by the payload data bytes. It is up to the application to determine the expected number of header bytes.
ISO 11898, SAE J1939	The first 4 bytes of the expected data are reserved for the CAN ID (11 bit or 29 bit).
SAE J2610	Expected message data from the vehicle serial bus. Expected response data does not include any echoed bytes when the protocol is in half-duplex mode.
SAE J1708	Expected data contains MID (byte 1) followed by any number of expected payload data bytes.

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

## Annex A (normative)

### D-PDU API compatibility mappings

#### A.1 Mapping of D-PDU API and D-Server API

Table A.1 — Mapping of D-PDU API send/receive cycles and D-Server API

Runtime mode	Repetition mode	NumSendCycles	NumReceiveCycles
eNONCYCLIC	eSINGLE	1	1
eNONCYCLIC	eREPEATED	1	1
eCYCLIC	eSINGLE	1	-1

#### A.2 Mapping of D-PDU API and ODX

Table A.2 — Mapping of D-PDU API send/receive cycles and ODX

ODX	NumSendCycles	NumReceiveCycles
IS-CYCLIC	1	-1
IS-MULTIPLE	1	-2

## Annex B (normative)

### D-PDU API standard ComParams and protocols

#### B.1 Standardized protocols — Support and naming conventions

##### B.1.1 General

The D-PDU API is not restricted to specific diagnostic protocols. Since the supported protocols and its ComParams are described in the MDF, the protocol support of an MVCI protocol module using the D-PDU API can be extended easily. The only important requirement is that the designation of the protocols (e.g. protocol names) is unique. To assure this requirement, see B.1.4. An attachment containing ComParams is available as an electronic file accompanying this document.

##### B.1.2 SAE J2534 and RP1210a standard protocol names

**Table B.1 — SAE J2534-1 and RP1210a standard protocol names**

Protocol name	Description
ISO_11898_RAW	Raw CAN protocol (layer 2); behaviour identical to protocol ID CAN in SAE J2534-1 or protocol string CAN in RP1210a.
ISO_15765_2	ISO 15765 protocol with ISO 15765-2 flow control enabled; behaviour identical to protocol ID ISO 15765 in SAE J2534-1.
ISO_15765_3	ISO 15765 protocol with automatic flow control handling, Tester Present handling and enhanced support of the ComLogicalLink concept. All address information (CAN identifiers) for physical and functional addressing is defined as ComParams for the ComLogicalLink. The application only needs to care for the ServiceID and data during communication.
ISO_14229_3	This protocol is the successor of ISO_15765_3 and implements the identical features and functions.
SAE_J1850_VPW	GM/DaimlerChrysler CLASS2; behaviour identical to protocol ID SAE J1850_VPW in SAE J2534-1 or protocol string SAE J1850 in RP1210a (using a SAE J1850_VPW MVCI protocol module).
SAE_J1850_PWM	Ford SCP; behaviour identical to protocol ID SAE J1850_PWM in SAE J2534-1 or protocol string SAE J1850 in RP1210a (using a SAE J1850_PWM MVCI protocol module).
SAE_J1939_21	SAE J1939 network protocol; behaviour identical to protocol string SAE J1939 in RP1210a.
SAE_J1708	SAE J1708 network protocol; behaviour identical to protocol string SAE J1708 in RP1210a.
SAE_J2610_SCI	SAE J2610 protocol (DaimlerChrysler SCI); four configurations are defined in protocol: configuration A for engine (ID: SCI_A_ENGINE), configuration A for transmission (ID: SCI_A_TRANS), configuration B for engine (ID: SCI_B_ENGINE) and configuration B for transmission (ID: SCI_B_TRANS).
ISO_14230_4	ISO 14230-4 (Keyword protocol 2000); behaviour identical to protocol ID ISO 14230 in SAE J2534-1.
ISO_9141_2	Raw ISO 9141 or ISO 9141-2 protocol; behaviour identical to protocol ID ISO 9141 in SAE J2534-1.

NOTE A specific MVCI protocol module is not required to support a minimum set of protocols. The protocols supported by the specific MVCI protocol module can be evaluated by the application using the MDF and the function call `PDUGetResourceStatus()`.

##### B.1.3 Protocol names — Combination list

Some protocols consist of several protocol layers which may be combined. All combinations shall be treated as a protocol definition itself, as it is not transparent to the application that is implemented in the D-PDU API.

The application layer ISO\_14229\_3 is the successor of ISO\_15765\_3; its PDU-API functionality is identical to ISO\_15765\_3. This behaviour is outlined in Table B.2 by referencing the same protocol full name "ISO UDS on CAN". Newer ISO UDS on CAN applications on top of PDU-API shall use ISO\_14229\_3 instead of ISO\_15765\_3 for the application layer. ISO\_15765\_3 within this specification is only mentioned for backward

compatibility reasons to support existing applications which still reference the ISO\_15765\_3 application layer protocol. MVCIs implementing PDU-API shall treat both application protocols the same in their implementation, meaning to provide the identical behaviour from application point of view when using ISO\_15765\_3 or ISO\_14229\_3 as the application layer protocol.

Table B.2 indicates the necessary layering of some standard protocols.

**Table B.2 — Standard protocol combination list**

Protocol full name	Application layer	Transport layer (CAN), DataLink layer (K-Line)	Physical layer(s)
KWP2000 on K Line	ISO_14230_3	ISO_14230_2	ISO_14230_1_UART
KWP2000 on CAN	ISO_14230_3	ISO_15765_2	ISO_11898_2_DWCAN, ISO_11898_3_DWFTCAN, SAE_J2411_SWCAN
ISO UDS on CAN	ISO_15765_3	ISO_15765_2	ISO_11898_2_DWCAN, ISO_11898_3_DWFTCAN, SAE_J2411_SWCAN
ISO UDS on CAN	ISO_14229_3	ISO_15765_2	ISO_11898_2_DWCAN, ISO_11898_3_DWFTCAN, SAE_J2411_SWCAN
Enhanced Diagnostics on ISO 14230 K-Line	SAE_J2190	ISO_14230_2	ISO_14230_1_UART
Enhanced Diagnostics on ISO 9141 K-Line	SAE_J2190	ISO_9141_2	ISO_9141_2_UART
Enhanced Diagnostics on CAN	SAE_J2190	ISO_15765_2	ISO_11898_2_DWCAN, ISO_11898_3_DWFTCAN, SAE_J2411_SWCAN
Enhanced Diagnostics on SAE J1850_VPW	SAE_J2190	SAE_J1850_VPW	SAE_J1850_VPW
Enhanced Diagnostics on SAE J1850_PWM	SAE_J2190	SAE_J1850_PWM	SAE_J1850_PWM
ISO OBD on K-Line	ISO_15031_5	ISO_9141_2 ISO_14230_4	ISO_9141_2_UART ISO_14230_1_UART
ISO OBD on SAE J1850	ISO_15031_5	SAE_J1850_VPW SAE_J1850_PWM	SAE_J1850_VPW SAE_J1850_PWM
ISO OBD on CAN	ISO_15031_5	ISO_15765_4	ISO_11898_2_DWCAN
ISO RAW CAN	ISO_11898_RAW	N/A	ISO_11898_2_DWCAN, ISO_11898_3_DWFTCAN, SAE_J2411_SWCAN
Truck and Bus on CAN	SAE_J1939_73	SAE_J1939_21	SAE_J1939_11_DWCAN
Truck and Bus on UART	SAE_J1587	SAE_J1708	SAE_J1708_UART
Chrysler SCI	SAE_J2610	SAE_J2610_SCI	SAE_J2610_UART
Enhanced Diagnostics on ISO 9141 K-Line and ISO 14230 K-Line	SAE_J2190	ISO_9141_2_and_ISO_14230_2	ISO_9141_2_UART_and_ISO_14230_1_UART
ISO OBD on K-Line	ISO_15031_5	ISO_9141_2 ISO_14230_4	ISO_9141_2_UART ISO_14230_1_UART ISO_9141_2_UART_and_ISO_14230_1_UART

Table B.2 — (continued)

Protocol full name	Application layer	Transport layer (CAN), DataLink layer (K-Line)	Physical layer(s)
Enhanced Diagnostics on SAE J1850	SAE_J2190	SAE_J1850	SAE_J1850
ISO OBD on SAE J1850	ISO_15031_5	SAE_J1850_VPW SAE_J1850_PWM	SAE_J1850_VPW SAE_J1850_PWM SAE_J1850
ISO UDS on DoIP	ISO_14229_5(UDS)	ISO_13400_2(DoIP)	IEEE_802_3(Ethernet)

#### B.1.4 Standard protocol naming guidelines

The following naming guidelines apply.

- If the protocol is specified as a standard, the designation of the standard shall be used and the standardization organization shall be included in the name as a prefix (e.g. ISO\_99999, SAE\_J8888). If the protocol is specified as part of a standard, the part number shall be included in the name as postfix (e.g. ISO\_15765\_2).
- The standard protocol short names which are used in the MDF file for the PROTOCOL element are a concatenation of the application layer specification name, plus the transport layer specification layer name, connected by the additional string “\_on\_”, as shown in Tables B.3 and B.4.
- The physical layer name as shown in Table B.2 is used in the MDF file as short name for the BUSTYPE element.

#### B.1.5 Standard protocol short names

Table B.3 — Standard protocol short names in ODX

Short name	Protocol description
ISO_14230_3_on_ISO_14230_2	KWP2000 on K-Line
ISO_14230_3_on_ISO_15765_2	KWP2000 on CAN
ISO_15765_3_on_ISO_15765_2	ISO UDS on CAN (same as ISO_14229_3_on_ISO_15765_2)
ISO_14229_3_on_ISO_15765_2	ISO UDS on CAN (same as ISO_15765_3_on_ISO_15765_2)
SAE_J2190_on_ISO_14230_2	Enhanced Diagnostics on KWP2000 K-Line
SAE_J2190_on_ISO_9141_2	Enhanced Diagnostics on 9141 K-Line
SAE_J2190_on_ISO_15765_2	Enhanced Diagnostics on CAN
SAE_J2190_on_SAE_J1850_VPW	Enhanced Diagnostics on SAE J1850_VPW
SAE_J2190_on_SAE_J1850_PWM	Enhanced Diagnostics on SAE J1850_PWM
ISO_15031_5_on_ISO_9141_2	ISO OBD on 9141-2 K-Line
ISO_15031_5_on_SAE_J1850_VPW	ISO OBD on SAE J1850 VPW
ISO_15031_5_on_ISO_15765_4	ISO OBD on CAN
ISO_15031_5_on_SAE_J1850_PWM	ISO OBD on SAE J1850 PWM
ISO_15031_5_on_ISO_14230_4	ISO OBD on KWP2000 K-Line
ISO_15031_5_on_SAE_J1939_73	ISO OBD on Truck and Bus CAN
SAE_J1939_73_on_SAE_J1939_21	Truck and Bus on CAN
SAE_J1587_on_SAE_J1708	Truck and Bus on UART
SAE_J2610_on_SAE_J2610_SCI	Chrysler SCI
ISO_11898_RAW	ISO RAW CAN

Table B.3 — (continued)

Short name	Protocol description
SAE_J2190_on_ISO_9141_2_and_ISO_14230_2	Enhanced Diagnostics on 9141 K-Line and KWP2000 K-Line
ISO_15031_5_on_ISO_9141_2_and_ISO_14230_4	ISO OBD on 9141-2 K-Line and KWP2000 K-Line
SAE_J2190_on_SAE_J1850	Enhanced Diagnostics on SAE J1850
ISO_15031_5_on_SAE_J1850	ISO OBD on SAE J1850
ISO_14229_5_on_ISO_13400_2	ISO UDS on DoIP

Since some of the default values differ between the two supported bus types, it is not possible to define a single default value. If the definition of a single default value is mandatory (e.g. in ODX COMPARAM-SPECs), the value shall be ignored by the D-PDU API. Implementations shall set the values of those ComParams internally according to Table J.2 after detection of the used bus type.

### B.1.6 D-PDU API optional OBD protocol short names

These OBD protocols will allow an OBD application to perform a quick OBD initialization. Many features of an OBD initialization are supported internally by the D-PDU API when the below protocols are supported. See Annex J for more information regarding OBD initialization supported by the D-PDU API.

Table B.4 — OBD protocol shortnames

Short name	Protocol description
ISO_OBD_on_K_Line	ISO OBD on 9141-2 K-Line and KWP2000 K-Line
ISO_OBD_on_SAE_J1850	ISO OBD on SAE J1850 VPW and SAE J1850 PWM
ISO_OBD_on_ISO_15765_4	ISO OBD on CAN
ISO_OBD_on_SAE_J1939_73	ISO OBD on Truck and Bus CAN

## B.2 Standard protocol pin types and short names

The MVCI protocol module communicates to a vehicle serial bus using one or more vehicle connector pins. These pins types can be referenced by their ODX compliant short-names in Table B.5.

Table B.5 — Pin type short names

Short name	Pin type description (example protocol usage)
HI	Differential Line - High (e.g. DW_CAN High)
LOW	Differential Line - Low (e.g. DW_CAN Low), DoIP TX_Minus_Line
K	UART K-Line (e.g. KWP2000)
L	UART L-Line (e.g. ISO 9141-2)
TX	UART uni-directional transmit (e.g. SAE J2190), DoIP TX_Plus_Line
RX	UART uni-directional receive (e.g. SAE J2190), DoIP RX_Plus_Line
PLUS	SAE J1850 Plus (e.g. SAE J1850 VPW and SAE J1850 PWM)
MINUS	SAE J1850 Minus (e.g. SAE J1850 PWM), DoIP RX_Minus_Line
SINGLE	Single wire [e.g. SW_CAN, and UART bi-directional transmit/receive (e.g. SAE J2740)]
PROGV	Pin to set the programmable voltage on DLC
IGNITION-CLAMP	Pin to read the ignition sense state from DLC

### B.3 Standard protocol communication parameters (ComParams)

#### B.3.1 Protocol ComParam description method

ISO 22901-1 (ODX specification) already defines mechanisms for the description of ComParams for a protocol. The D-PDU API can be used in combination with ODX data files. Therefore, the description mechanisms from the ODX specification are used to define ComParams in the MDF (for details, see the ODX specification).

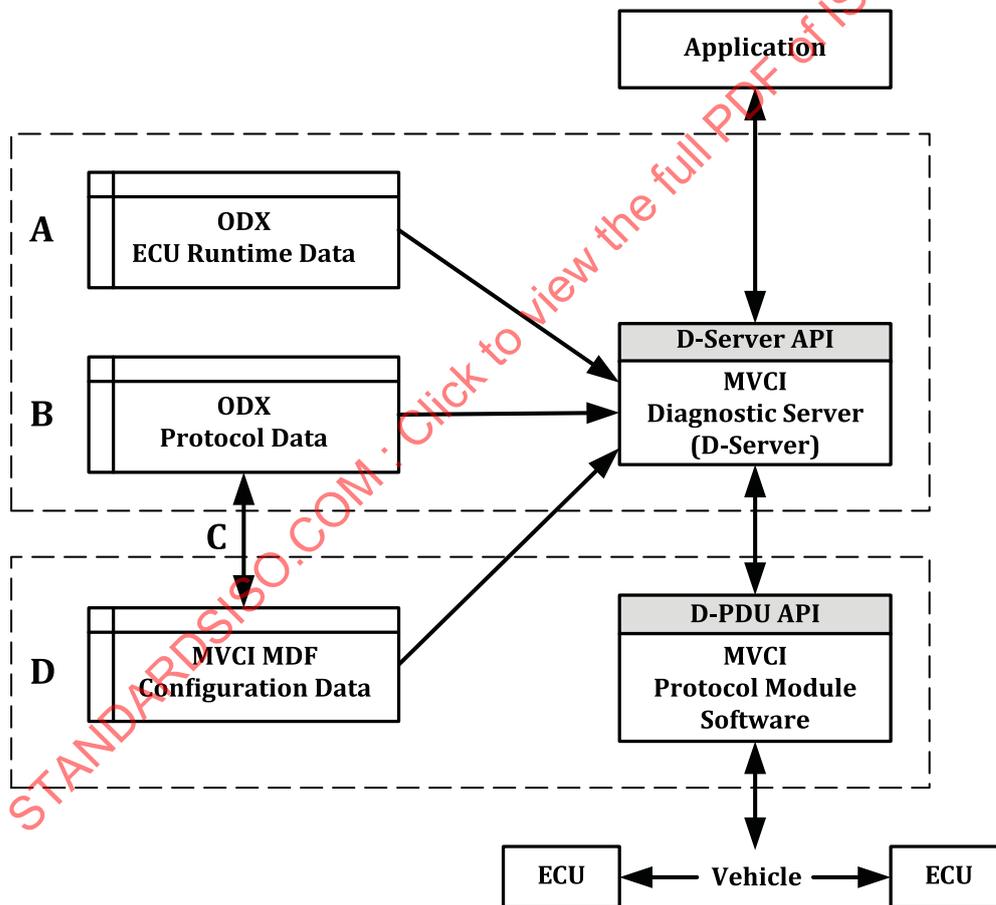
For each protocol that the MSCI protocol module supports, the MDF shall assign the reference between the ComParams and the unique protocol id. For each protocol, the MDF includes the following elements:

- ProtocolName;
- Short name and unique ID for each protocol ComParam.

The format for each standard protocol ComParam short name shall be CP\_yyyy, where yyyy is the parameter name. An example of a standard protocol ComParam short name would be CP\_Baudrate.

NOTE For protocol ComParams, only parameter data types supported by the D-PDU API are used.

Figure B.1 illustrates the relationship between the various definition files.



- Key**
- A ShortName=CP\_Baudrate, LongName=Baudrate for CAN bus, DataType=PDU\_PT\_UNUM32, Min=500000, Max=500000, Default=500000.
  - B File: ISO15765.ODX, describes ISO 15765 protocol capabilities.
  - C ComParams, protocol and bus type shall match between B and D.
  - D Protocol=ISO15765, ShortName=CP\_Baudrate, ID=0x1234.

Figure B.1 — Example for ComParam configuration

The application (in this case the MVCI D-Server) uses the ECU ODX file for a specific ECU, which uses protocol ISO 15765. This file contains the protocol ComParams, which are required for the specific ECU. For example, the CAN bus baud rate for the specific ECU is specified by the short name *Cp\_Baudrate*. The MDF contains entries for all protocol ComParams of each protocol supported by the MVCI protocol module. Therefore, it contains the entry *Cp\_Baudrate* for protocol ISO 15765 together with the assigned ID value of 0x1234. The ID value has been assigned by the MVCI protocol module supplier and it is documented in the MDF. The ID value can now be used in D-PDU API function calls.

The MDF contains the following information about a ComParam:

- short name;
- long name;
- ComParam class;
- layer Info;
- ComParam data type;
- minimum value;
- mvalue;
- default value (per protocol).

To avoid the usage of different short names for the same ComParams with MVCI protocol modules from different suppliers, see Tables B.1 and B.3. In most cases, the ComParam names are similar or even identical to those used in SAE J2534-1 and RP1210a specifications (see these specifications for details). For different protocols, it is recommended to use the same designations for protocol ComParams in the corresponding protocol specifications. It is not required that a MVCI protocol module support all listed protocol ComParams. However, if a MVCI protocol module supports a protocol ComParam listed below, it is mandatory that the listed protocol ComParam short name is used in the MDF.

The ID value for each protocol ComParam can be freely assigned by the MVCI supplier because the ID value is used only within the supplier-specific D-PDU API.

### B.3.2 ComParam class

Each ComParam belongs to an ODX PARAM-CLASS. Table B.6 includes a list of the relevant ODX PARAM-CLASS.

```
typedef enum E_PDU_PC {
    PDU_PC_TIMING      = 1,
    PDU_PC_INIT       = 2,
    PDU_PC_COM        = 3,
    PDU_PC_ERRHDL     = 4,
    PDU_PC_BUSTYPE    = 5,
    PDU_PC_UNIQUE_ID  = 6,
    PDU_PC_TESTER_PRESENT = 7,
} T_PDU_PC;
```

Table B.6 — Definition of the ODX ComParam classes

D-PDU API ComParam Class	ODX PARAM-CLASS	Class description
PDU_PC_TIMING	TIMING	Message flow timing ComParams, e.g. inter-byte time or time between request and response.
PDU_PC_INIT	INIT	ComParams for initiation of communication, e.g. trigger address or wakeup pattern.
PDU_PC_COM	COM	General communication ComParam.
PDU_PC_ERRHDL	ERRHDL	ComParam defining the behaviour of the runtime system in case an error occurred, e.g. runtime system could either continue communication after a timeout was detected or stop and reactivate.
PDU_PC_BUSTYPE	BUSTYPE	This is used to define a bustype specific ComParam (e.g. baud rate). Most of these ComParams affect the physical hardware. These ComParams can only be modified by the first ComLogicalLink that acquired the physical resource (PDUCreateComLogicalLink()). When a second ComLogicalLink is created for the same resource, these ComParams that were previously set by the initial ComLogicalLink will be active for the new ComLogicalLink.
PDU_PC_UNIQUE_ID	UNIQUE_ID	This type of ComParam is used to indicate to both the ComLogicalLink and the application that the information is used for protocol response handling from a physical or functional group of ECUs to uniquely define an ECU response.
PDU_PC_TESTER_PRESENT	TESTER_PRESENT	This type of ComParam is used for tester present type of ComParams (CP_TesterPresentxxx). Tester Present ComParams cannot be changed temporarily using the TempParamUpdate flag like other ComParams. Using this type of ComParam class enables an application and database to properly configure and use Tester Present ComParams.

### B.3.3 ComParam data type

Each ComParam has a data type which describes the possible data types for ComParams.

```
typedef enum E_PDU_PT {
    PDU_PT_UNUM8      = 0x00000101,      /* Unsigned byte */
    PDU_PT_SNUM8      = 0x00000102,      /* Signed byte */
    PDU_PT_UNUM16     = 0x00000103,      /* Unsigned two bytes */
    PDU_PT_SNUM16     = 0x00000104,      /* Signed two bytes */
    PDU_PT_UNUM32     = 0x00000105,      /* Unsigned four bytes */
    PDU_PT_SNUM32     = 0x00000106,      /* Signed four bytes */
    PDU_PT_BYTEFIELD = 0x00000107,      /* Structure contains an array of UNUM8 bytes with a maximum
                                         length and actual length fields. See B.3.3.1 for the definition. */
    PDU_PT_STRUCTFIELD = 0x00000108,     /* Structure contains a void * pointer to an array of structures. The
                                         ComParamStructType item determines the type of structure to be
                                         typecasted onto the void * pointer. This structure contains a field
                                         for maximum number of struct entries and the actual number of
                                         struct entries. See B.3.3.2 for the definition. */
    PDU_PT_LONGFIELD  = 0x00000109,     /* Structure contains an array of UNUM32 entries with a maximum
                                         length and actual length fields. See B.3.3.3 for the definition. */
} T_PDU_PT;
```

#### B.3.3.1 ComParam BYTEFIELD data type

```
typedef struct {
    UNUM32 ParamMaxLen; /* Contains the maximum number of UNUM8 bytes the ComParam can contain in
```

pdataArray. This is also the amount of memory the D-PDU API allocates prior to a call of PDUGetComParam. The value of ParamMaxLen is given in the MDF file as part of the DEFAULT\_VALUE entry of the corresponding COMPARAM\_REF or COMPARAM (see F.2.2). \*/

NOTE 1 A MAX\_VALUE entry does not appear at the corresponding COMPARAM\_REF or COMPARAM for this type of ComParam.

UNUM32 ParamActLen;

/\* Contains the actual number of UNUM8 bytes in pdataArray. The value of ParamActLen is given in the MDF file as part of the DEFAULT\_VALUE entry of the corresponding COMPARAM\_REF or COMPARAM (see F.2.2). \*/

NOTE 2 A MIN\_VALUE entry does not appear at the corresponding COMPARAM\_REF or COMPARAM for this type of ComParam.

UNUM8 \*pdataArray;

/\* Pointer to an array of UNUM8 values \*/

} PDU\_PARAM\_BYTEFIELD\_DATA;

NOTE 3 In the MDF (see F.2), the elements MIN\_VALUE, MAX\_VALUE and DEFAULT\_VALUE are optional at the COMPARAM\_REF and COMPARAM, except that only DEFAULT\_VALUE is mandatory at COMPARAM. This is to ensure that ParamMaxLen is always retrievable from the MDF.

### B.3.3.2 ComParam STRUCTFIELD data type

typedef struct {

T\_PDU\_CPST ComParamStructType;

/\* type of ComParam Structure being used. See B.3.3.2.1\*/

UNUM32 ParamMaxEntries;

/\* Contains the maximum number of struct entries the ComParam can contain in pStructArray. The D-PDU API allocates this amount of memory based on the size of the structure type prior to a call of PDUGetComParam. The value of ParamMaxLen is given in the MDF file as part of the DEFAULT\_VALUE entry of the corresponding COMPARAM\_REF or COMPARAM (see F.2.2). \*/

NOTE 1 A MAX\_VALUE entry does not appear at the corresponding COMPARAM\_REF or COMPARAM for this type of ComParam.

UNUM32 ParamActEntries;

/\* Contains the actual number of struct entries in pStructArray. The value of ParamActLen is given in the MDF file as part of the DEFAULT\_VALUE entry of the corresponding COMPARAM\_REF or COMPARAM (see F.2.2). \*/

NOTE 2 A MIN\_VALUE entry does not appear at the corresponding COMPARAM\_REF or COMPARAM for this type of ComParam.

void \*pStructArray;

/\* Pointer to an array of structs (typecasted to the ComParamStructType) \*/

} PDU\_PARAM\_STRUCTFIELD\_DATA;

NOTE 3 STRUCTFIELD type structures (i.e. structures pointed to by pStructArray) are on even byte boundaries.

NOTE 4 In the MDF (see F.2), the elements MIN\_VALUE, MAX\_VALUE and DEFAULT\_VALUE are optional at the COMPARAM\_REF and COMPARAM, except that only DEFAULT\_VALUE is mandatory at COMPARAM. This is to ensure that ParamMaxLen is always retrievable from the MDF.

#### B.3.3.2.1 ComParamStructType typedef

Each ComParam of type PDU\_PT\_STRUCTFIELD has a type (ComParamStructType) which describes the structure expected in the ComParam. The typedef T\_PDU\_CPST is used in this structure.

typedef enum E\_PDU\_CPST{

PDU\_CPST\_SESSION\_TIMING = 0x00000001, /\* See ComParam struct type

PDU\_PARAM\_STRUCT\_SESS\_TIMING \*/

PDU\_CPST\_ACCESS\_TIMING = 0x00000002, /\* See ComParam struct type

PDU\_PARAM\_STRUCT\_ACCESS\_TIMING

} T\_PDU\_CPST;

**B.3.3.2.2 ComParam STRUCTFIELD = Session timing**

Structure used for a STRUCTFIELD ComParam of ComParamStructType = PDU\_CPST\_SESSION\_TIMING.

```
typedef struct {
    UNUM16 session;          /* Session Number, for the diagnostic session of ISO 15765-3 / ISO14229-3 */
    UNUM8 P2Max_high;       /* 1 ms resolution, High byte of the default P2Can_Server_max timing supported by the
                             server for the activated diagnostic session. Used for ComParam CP_P2Max. */
    UNUM8 P2Max_low;        /* 1 ms resolution. Low byte of the default P2Can_Server_max timing supported by the
                             server for the activated diagnostic session. Used for ComParam CP_P2Max. */
    UNUM8 P2Star_high;      /* 10 ms resolution. High byte of the enhanced (NRC 78 hex) P2Can_Server_max
                             supported by the server for the activated diagnostic session. Used for ComParam
                             CP_P2Star */
    UNUM8 P2Star_low;       /* 10 ms resolution. Low byte of the enhanced (NRC 78 hex) P2Can_Server_max supported
                             by the server for the activated diagnostic session. Used for ComParam CP_P2Star. */
} PDU_PARAM_STRUCT_SESS_TIMING;
```

NOTE The valid value range for the struct member "session" is [1;127] ([0x01;0x7F]) only.

In ODX COMPARAM-SPECs requiring the definition of a PHYSICAL-DEFAULT-VALUE for each member of the defined structure, these default values shall be set to 0.

**B.3.3.2.3 ComParam STRUCTFIELD = Access timing**

Structure used for a STRUCTFIELD ComParam of ComParamStructType = PDU\_CPST\_ACCESS\_TIMING. This structure is used for both ECU ComParams and Tester ComParams (see CP\_AccessTiming\_Ecu and CP\_AccessTimingOverride).

```
typedef struct {
    UNUM8 P2Min;            /* 0,5 ms resolution. Minimum time between tester request and ECU response(s). Used for
                             ComParam CP_P2Min */
    UNUM8 P2Max;            /* Resolution (see ISO 14230-2:2013, Table 5). Maximum time between tester request and ECU
                             response(s). Used for ComParam CP_P2Max. */
    UNUM8 P3Min;            /* 0,5 ms resolution. Minimum time between end of ECU responses and start of new tester
                             request. Used for ComParam CP_P3Min. */
    UNUM8 P3Max;            /* 250 ms resolution. Maximum time between ECU responses and start of new tester request
                             Used for ComParam CP_P3Max_Ecu or CP_P2Star for the Tester */
    UNUM8 P4Min;            /* 0,5 ms resolution. Minimum inter byte time for tester request. Used for ComParam CP_P4Min.
                             */
    UNUM8 TimingSet;        /* Set number allowing multiple sets of timing parameters:
                             1 = default timing set used by ECU on a TPI 1 request from the Tester. (Values also returned by
                             the ECU on a TPI 2.)
                             2 = override timing values received by ECU (Tester does not use values returned by ECU, but
                             instead uses values in this structure. (Used during a positive TPI 2 response))
                             3 = override timing values set by Tester (ECU does not use values set by Tester, but instead
                             uses values in this structure. (Used during a request of TPI 3.))
                             4 = normal timing values used by the ECU after initialization.
                             0xFF = extended timing (values in this structure are used after a keyword initialization (if
                             extended timing is supported by the key bytes). Timing is used by both tester and ECU) */
} PDU_PARAM_STRUCT_ACCESS_TIMING;
```

In ODX COMPARAM-SPECs requiring the definition of a PHYSICAL-DEFAULT-VALUE for each member of the defined structure, these default values shall be set to 0.

### B.3.3.3 ComParam LONGFIELD Data Type

```
typedef struct {
    UNUM32    ParamMaxLen;    /* Contains the maximum number of UNUM32 entries the ComParam can contain
                               in pDataArray. The D-PDU API allocates this amount of UNUM32 memory prior to
                               a call of PDUGetComParam. The value of ParamMaxLen is given in the MDF file
                               as part of the DEFAULT_VALUE entry of the corresponding COMPARAM_REF or
                               COMPARAM (see F.2.2). */
    NOTE 1    A MAX_VALUE entry does not appear at the corresponding
    COMPARAM_REF or COMPARAM for this type of ComParam.
    UNUM32    ParamActLen;    /* Contains the actual number of UNUM32 entries in pDataArray. The value of
                               ParamActLen is given in the MDF file as part of the DEFAULT_VALUE entry of
                               the corresponding COMPARAM_REF or COMPARAM (see F.2.2). */
    NOTE 2    A MIN_VALUE entry does not appear at the corresponding
    COMPARAM_REF or COMPARAM for this type of ComParam.
    UNUM32    *pDataArray;    /* Pointer to an array of UNUM32 values */
} PDU_PARAM_LONGFIELD_DATA;
```

NOTE 3 In the MDF (see F.2), the elements MIN\_VALUE, MAX\_VALUE and DEFAULT\_VALUE are optional at the COMPARAM\_REF and COMPARAM, except that only DEFAULT\_VALUE is mandatory at COMPARAM. This is to ensure that ParamMaxLen is always retrievable from the MDF.

### B.3.4 ComParam support

Each ComParam may or may not be supported by a MSCI protocol module. If a MSCI protocol module supports a protocol, then only the ComParams that are considered “standard” shall be supported by the MSCI protocol module. OEM-specific ComParams may be defined but will not be part of the standard.

ECU specific ComParams will not be interpreted by the D-PDU API. In case an ECU-specific ComParam is passed to the D-PDU API, the API will return an error code.

**Table B.7 — Definition of the ODX ComParam support types**

ComParam support	Acronym	Type description
PDU_PS_STANDARD	S	The ComParam belonging to a standardized protocol has to be supported by the D-PDU API system to be compliant with this document for every protocol supported by the D-PDU API. These ComParams apply to both the tester and the ECU unless CPUSAGE (see B.3.5) is specified.
PDU_PS_OPTIONAL	O	This ComParam does not have to be supported by the D-PDU API.

### B.3.5 ComParam usage

In the D-PDU API ComParam tables, the optional ComParam usage acronyms “T” and “E” are used to indicate whether a ComParam is relevant only for a Tester/MSCI protocol module (“T”) or whether it is only relevant for ECU software generation or configuration (“E”). If a ComParam is relevant for both tester and ECU, none of the acronyms “T” or “E” are used.

In ODX for each COMPARAM, the attribute CPUSAGE shall be set to one of the possible values: “TESTER”, “ECU-SOFTWARE”, “ECU-COMM” or “APPLICATION”. The relation between the CPUSAGE attribute used in ODX and the optional ComParam usage acronyms “T” and “E” used in the PDU API ComParam tables is shown in Table B.8.

Table B.8 — Definition of the ODX ComParam usage

ODX CPUSAGE attribute	ComParam usage acronym used in D-PDU API ComParam tables	Description
TESTER	T	The ComParam is specific to the tester (VCI) and is neither supported nor implemented by the ECU.
ECU-SOFTWARE	E	The ComParam is specific to the ECU and is neither supported nor implemented by the tester (VCI).
ECU-COMM	—	The ComParam is needed for communication between tester and ECU and shall be supported by both tester and ECU (e.g. addresses).
APPLICATION	N/A	The ComParam is only evaluated by the application. It is never passed down to the D-PDU API by an MCD 3D/MVCI Diagnostic Server. The ComParam is not specified within the D-PDU API standard.

### B.3.6 ComParam OSI layer reference

Each ComParam is applicable to a specific OSI Layer.

Table B.9 — Definition of the OSI layer

Layer information	Acronym	Layer description
PDU_PL_APPLICATION	APP	ComParams in this layer apply to timing and error handling elements that are referenced in the application layer specifications.
PDU_PL_TRANSPORT	TRANS	ComParams in this layer apply to timing and configuration elements that are referenced in the Transport/Network layer specifications. These ComParams are used to configure header bytes, develop CAN ids, framing information, checksums, etc.
PDU_PL_PHYSICAL	PHYS	ComParams in this layer apply to elements that are referenced in the datalink/physical layer specifications. These ComParams affect the physical characteristics of bus configuration.

## B.4 ComParam summary tables

### B.4.1 Application layer

Table B.10 lists the ComParams applicable to the application layer. If a ComParam is used by a particular protocol, the appropriate acronyms for the ODX ComParameter type (S, O) and the ODX ComParameter usage (T, E) are included.

Table B.10 — Application layer ComParam summary table

Parameter short name	Emissions OBD										ISO_15765.3/ ISO_14229_3	ISO_11898_RAW	SAE_11939_73	SAE_11587	SAE_12190	SAE_12610	ISO_14229_5	PARAM-CLASS
	ISO_15765_4	ISO_14230_4	ISO_9141_2	SAE_11850_VPW	SAE_11850_PWM	ISO_14230_3	ISO_15765.3/ ISO_14229_3	ISO_11898_RAW	SAE_11939_73	SAE_11587								
CP_CanTransmissionTime																		TIMING
CP_ChangeSpeedCtrl	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_ChangeSpeedMessage	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_ChangeSpeedRate	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_ChangeSpeedResCtrl	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_ChangeSpeedTxDelay	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	TIMING
CP_CyclicRespTimeout	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	TIMING
CP_DoIPConnectionCloseDelay																		TIMING
CP_EnablePerformanceTest	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_EnableInitSeqRepetition	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_Loopback	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM
CP_MessageIndicationRate																		TIMING
CP_ModifyTiming		O,T								O,T	O,T						O,T	TIMING
CP_NetworkTransmissionTime																		TIMING
CP_P2Max	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	TIMING
CP_P2Max_Ecu	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	S,E	TIMING
CP_P2Min	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	TIMING

Table B.10 — (continued)

Parameter short name	Emissions OBD								ISO_14230_3	ISO_15765_3/ ISO_14229_3	ISO_11898_RAW	SAE_11939_73	SAE_11587	SAE_12190	SAE_12610	ISO_14229_5	PARAM-CLASS
	ISO_15765_4	ISO_14230_4	ISO_9141_2	SAE_11850_VPW	SAE_11850_PWM	ISO_14230_3	ISO_15765_3/ ISO_14229_3	ISO_11898_RAW									
CP_P2Star	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T				TIMING
CP_P2Star_Ecu	S,E						S,E	S,E								S,E	TIMING
CP_P3Func									S,T	O,T						S,T	TIMING
CP_P3Max_Ecu		S,E	S,E				S,E										TIMING
CP_P3Min		S	S				S										TIMING
CP_P3Phys									S,T	O,T						S,T	TIMING
CP_P6Max																S,T	TIMING
CP_P6Star																S,T	TIMING
CP_RC21CompletionTimeout	S,T	S,T		S,T	S,T		S,T	S,T	S,T				S,T			S,T	ERRHDL
CP_RC21Handling	S,T	S,T		S,T	S,T		S,T	S,T	S,T				S,T			S,T	ERRHDL
CP_RC21RequestTime	S,T	S,T		S,T	S,T		S,T	S,T	S,T				S,T			S,T	ERRHDL
CP_RC23CompletionTimeout	O,T								O,T							O,T	ERRHDL
CP_RC23Handling	O,T								O,T							O,T	ERRHDL
CP_RC23RequestTime	O,T								O,T							O,T	ERRHDL
CP_RC78CompletionTimeout	S,T	S,T		S,T	S,T		S,T	S,T	S,T				S,T			S,T	ERRHDL
CP_RC78Handling	S,T	S,T		S,T	S,T		S,T	S,T	S,T				S,T			S,T	ERRHDL
CP_RCByteOffset	S,T	S,T	S,T	S,T	S,T		S,T	S,T	S,T				S,T			S,T	ERRHDL
CP_RepeatReqCountApp	O,T	O,T	O,T	O,T	O,T		O,T	O,T	O,T	O,T			O,T	O,T	O,T	O,T	ERRHDL
CP_SessionTiming_Ecu									S,E							S,E	TIMING
CP_SessionTimingOverride									O,T							O,T	TIMING

Table B.10 — (continued)

Parameter short name	Emissions OBD										PARAM-CLASS			
	ISO_15765_4	ISO_14230_4	ISO_9141_2	SAE_J1850_VPW	SAE_J1850_PWM	ISO_14230_3	ISO_15765_3/ ISO_14229_3	ISO_11898_RAW	SAE_J1939_73	SAE_J1587		SAE_J2190	SAE_J2610	ISO_14229_5
CP_StartMsgIndEnable	S,T	S,T	S,T			S,T	S,T		S,T	S,T				COM
CP_SuspendQueueOnError	O,T	O,T	O,T	O,T	O,T	O,T	O,T		O,T		O,T		O,T	ERRHDL
CP_SwCan_HighVoltage						S,T	S,T	S,T						COM
CP_TesterPresentAddrMode	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T		S,T	TESTER_PRESENT
CP_TesterPresentExpPos Resp	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T		S,T	TESTER_PRESENT
CP_TesterPresentExpNeg Resp	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T		S,T	TESTER_PRESENT
CP_TesterPresentHandling	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T		S,T	TESTER_PRESENT
CP_TesterPresentMessage	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T		S,T	TESTER_PRESENT
CP_TesterPresentReqRsp	S,T	S,T	S,T	S,T	S,T	S,T	S,T				S,T		S,T	TESTER_PRESENT
CP_TesterPresentSendType	S,T	S,T	S,T	S,T	S,T	S,T	S,T						S,T	TESTER_PRESENT
CP_TesterPresentTime	S,T	S,T	S,T	S,T	S,T	S,T	S,T						S,T	TESTER_PRESENT
CP_TesterPresentTime_Ecu	S,E	S,E		S,E	S,E	S,E	S,E				S,E		S,E	TESTER_PRESENT
CP_TransmitIndEnable	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	COM

## B.4.2 Transport layer

Table B.11 lists the ComParams applicable to the transport layer. If a ComParam is used by a particular protocol, the appropriate acronyms for the ODX ComParameter type (S, O) and the ODX ComParameter usage (T, E) are included.

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

Table B.11 — Transport layer ComParam summary table

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_19441_2	ISO_15765_2	ISO_14230_2	SAE_1939_21	SAE_1708	SAE_1850_VPW	SAE_1850_PWM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_5BaudAddressInverted			O,T		O,T								COM
CP_5BaudAddressFunc	S		S		S								COM
CP_5BaudAddressPhys			S		S								COM
CP_5BaudMode			S,T		S,T								INIT
CP_5BaudCommBaudrateOverride			O,T		O,T								INIT
CP_5BaudInitBaudrate			O,T		O,T								INIT
CP_AccessTiming_Ecu	S,E				S,E								TIMING
CP_AccessTimingOverride	O,T				O,T								TIMING
CP_Ar	S,T												TIMING
CP_Ar_Ecu	S,E												TIMING
CP_As	S,T												TIMING
CP_As_Ecu	S,E												TIMING
CP_BlockSize	S,T					S,T	S,T						COM
CP_BlockSize_Ecu	S,E					S,E	S,E						COM
CP_BlockSizeOverride	O,T					O,T	O,T						COM

STANDARDSISO.COM: Click to view the full PDF of ISO 22900-2:2017

Table B.11 — (continued)

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_9141_2	ISO_15765_2	ISO_14230_2	SAE_J1939_21	SAE_J1708	SAE_J1850_VPW	SAE_J1850_PWM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_Br	O,T			O,T		S	S						TIMING
CP_Br_Ecu	S,E			S,E		S,E	S,E						TIMING
CP_Bs	S,T			S,T		S	S						TIMING
CP_Bs_Ecu	S,E			S,E		S,E	S,E						TIMING
CP_CanDataSizeOffset	O,T			O,T									COM
CP_CanFillerByte	S			S		S					S		COM
CP_CanFillerByteHandling	S			S		S					S		COM
CP_CanFirstConsecutive-FrameValue	O,T			O,T									COM
CP_CanFuncReqExtAddr	S			S									COM
CP_CanFuncReqFormat	S			S									COM
CP_CanFuncReqId	S			S									COM
CP_CanMaxNumWaitFrames	S			S		S							COM
CP_CanPhysReqExtAddr	S			S							S		UNIQUE_ID
CP_CanPhysReqFormat	S			S							S		UNIQUE_ID
CP_CanPhysReqId	S			S							S		UNIQUE_ID
CP_CanRespUSDTExtAddr	S			S									UNIQUE_ID
CP_CanRespUSDTFormat	S			S									UNIQUE_ID

Table B.11 — (continued)

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_9141_2	ISO_15765_2	ISO_14230_2	SAE_J1939_21	SAE_J1708	SAE_J1850_VPW	SAE_J1850_PWM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_CanRespUSDTId	S			S									UNIQUE_ID
CP_CanRespUUDTExtAddr	S			S							S		UNIQUE_ID
CP_CanRespUUDTFormat	S			S							S		UNIQUE_ID
CP_CanRespUUDTId	S			S							S		UNIQUE_ID
CP_CANFDTxMaxDataLength				O,T							O,T		COM
CP_Cr	S,T			S,T		S	S						TIMING
CP_Cr_Ecu	S,E			S,E		S,E	S,E						TIMING
CP-Cs	O,T			O,T		S	S						TIMING
CP-Cs_Ecu	S,E			S,E		S,E	S,E						TIMING
CP_DisableTransportChecksumCheck		0	O, T		O, T								COM
CP_DoIPLogicalGatewayAddress												S	COM
CP_DoIPLogicalTesterAddress												S	COM
CP_DoIPLogicalEcuAddress												S	UNIQUE_ID
CP_DoIPSecondaryLogicalECUResponseAddress												S	UNIQUE_ID

Table B.11 — (continued)

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_14141_2	ISO_15765_2	ISO_14230_2	SAE_J1939_21	SAE_J1708	SAE_J1850_VPW	SAE_J1850_PWM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_DoIPLogicalFunctionalAddress												S	COM
CP_DoIPNumberOfRetries												S,T	COM
CP_DoIPDiagnosticAckTimeout												S,T	TIMING
CP_DoIPRetryPeriod												S,T	TIMING
CP_DoIPRoutingActivationTimeout												S,T	COM
CP_DoIPRoutingActivationType												S,T	COM
CP_EcuRespSourceAddress	S	S	S		S			S	S				UNIQUE_ID
CP_EnableConcatenation	S,T	S,T	S,T		S,T			S,T	S,T				COM
CP_EscapeSequenceHandling				O,T									COM
CP_ExtendedTiming	S				S								TIMING
CP_FillerByte	S	S	S		S			S	S				COM
CP_FillerByteHandling	S	S	S		S			S	S				COM
CP_FillerByteLength	S	S	S		S			S	S				COM
CP_FuncReqFormatPriority-Type	S,T	S,T	S,T		S,T			S,T	S,T				COM
CP_FuncReqTargetAddr	S,T	S,T	S,T		S,T			S,T	S,T				COM
CP_FuncRespFormatPriority-Type	S	S	S		S			S	S				UNIQUE_ID

Table B.11 — (continued)

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_19441_2	ISO_15765_2	ISO_14230_2	SAE_J1939_21	SAE_J1708	SAE_J1850_VPW	SAE_J1850_PWM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_FuncRespTargetAddr	S	S	S	S	S				S				UNIQUE_ID
CP_HeaderFormat1850									S,T				COM
CP_HeaderFormatKW	S,T		S,T	S,T									COM
CP_InitializationsSettings	S	S	S	S									INIT
CP_ISOKeyByteCount			O,T	O,T									INIT
CP_J1939AddrClaimTimeout						S							TIMING
CP_J1939AddressNegotiation-Rule						S							COM
CP_J1939DataPage						S							COM
CP_J1939MaxPacketTx						S							COM
CP_J1939Name						S,T							INIT
CP_J1939Name_Ecu						SE							INIT
CP_J1939PDUFormat						S							COM
CP_J1939PDUSpecific						S							COM
CP_J1939PreferredAddress						S,T							INIT
CP_J1939PreferredAddress_Ecu						SE							INIT
CP_J1939SourceAddress						S,T							UNIQUE_ID
CP_J1939SourceName						S,T							UNIQUE_ID
CP_J1939TargetAddress						S							COM

Table B.11 — (continued)

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_9141_2	ISO_15765_2	ISO_14230_2	SAE_J1939_21	SAE_J1708	SAE_J1850_VPW	SAE_J1850_PWPM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_J1939TargetName						S							COM
CP_MaxDataLength_Ecu				O,E									COM
CP_MessagePriority						S	S						COM
CP_MidReqId							S						COM
CP_MidRespId							S						UNIQUE_ID
CP_P1Max		S	S		S								TIMING
CP_P1Min		O	O		O								TIMING
CP_P4Max		O	O		O								TIMING
CP_P4Min		S	S		S								TIMING
CP_PhysReqFormatPriority-Type		S,T	S,T		S,T			S,T	S,T				COM
CP_PhysReqTargetAddr		S	S		S			S	S				COM
CP_PhysRespFormatPriority-Type		S	S		S			S	S				UNIQUE_ID
CP_RepeatReqCountTrans	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T	O,T		O,T	ERRHDL
CP_RequestAddrMode	S,T	S,T	S,T	S,T	S,T			S,T	S,T			S,T	COM
CP_SCITransmitMode										S			INIT
CP_SendRemoteFrame	S			S		S					S		COM
CP_StMin	S,T			S,T									TIMING
CP_StMin_Ecu	S,E			S,E									TIMING

Table B.11 — (continued)

Parameter short name	ISO_15765_4	ISO_14230_4	ISO_19411_2	ISO_15765_2	ISO_14230_2	SAE_J1939_21	SAE_J1708	SAE_J1850_VPW	SAE_J1850_PWM	SAE_2610_SCI	ISO_11898_RAW	ISO_13400_2	PARAM-CLASS
CP_StMinOverride	O,T			O,T									TIMING
CP_T1Max										S			TIMING
CP_T2Max										S			TIMING
CP_T3Max						S				S			TIMING
CP_T4Max						S				S			TIMING
CP_T5Max						S				S			TIMING
CP_TesterSourceAddress	S,T	S,T	S,T		S,T			S,T	S,T				COM
CP_TIdle	S	S	S		S								INIT
CP_TInil	S	S	S		S								INIT
CP_TPConnectionManagement							S,T						COM
CP_TWup	S		S		S								INIT
CP_W1Max	S	S	S		S								INIT
CP_W1Min	O	O	O		O								INIT
CP_W2Max	S	S	S		S								INIT
CP_W2Min	O	O	O		O								INIT
CP_W3Max	S	S	S		S								INIT
CP_W3Min	O	O	O		O								INIT
CP_W4Max	S	S	S		S								INIT
CP_W4Min	S	S	S		S								INIT

STANDARDSISO.COM · Click to view the full PDF of ISO 22900-2:2017

### B.4.3 Physical layer

Table B.12 lists the ComParams applicable to the physical layer. If a ComParam is used by a particular protocol, the appropriate acronyms for the ODX ComParameter type (S, O) and the ODX ComParameter usage (T, E) are included.

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

Table B.12 — Physical layer ComParam summary table

Parameter short name	ISO 11898_2_DWCAN	ISO 11898_3_DWFTCAN	ISO 11992_1_DWCAN	ISO 9141_2_UART	ISO 14230_1_UART	SAE J2610_UART	SAE J1708_UART	SAE J1939_11_DWCAN	SAE J1850_VPW	SAE J1850_PWM	SAE J2411_SWCAN	PARAM-CLASS
CP_Baudrate	S	S	S	S	S	S	S	S	S	S	S	BUSTYPE
CP_CANFDBaudrate	O,T											BUSTYPE
CP_BitSamplePoint	S,T	S,T	S,T					S,T			S,T	BUSTYPE
CP_BitSamplePoint_Ecu	S,E	S,E	S,E					S,E			S,E	BUSTYPE
CP_CANFDBitSamplePoint	O,T											BUSTYPE
CP_CANFDBitSamplePoint_Ecu	O,E											BUSTYPE
CP_CanBaudrateRecord	O,T	O,T	O,T					O,T				BUSTYPE
CP_K_L_LineInit				O								BUSTYPE
CP_K_LinePullup				O	O							BUSTYPE
CP_ListenOnly	O	O	O					O			O	BUSTYPE
CP_NetworkLine										S		BUSTYPE
CP_SyncJumpWidth	S,T	S,T	S,T					S,T			S,T	BUSTYPE
CP_SyncJumpWidth_Ecu	S,E	S,E	S,E					S,E			S,E	BUSTYPE
CP_CANFDSyncJumpWidth	O,T											BUSTYPE
CP_CANFDSyncJumpWidth_Ecu	O,E											BUSTYPE
CP_TerminationType	O,T							O,T			O,T	BUSTYPE
CP_TerminationType_Ecu											O,E	BUSTYPE
CP_UartConfig				S	S	S	S					BUSTYPE

#### B.4.4 CAN identifier format for ISO 15765 and ISO 11898 protocols

The protocols transport layer uses the CAN identifier format information to properly format CAN frames and to construct the correct CAN ID. For received frames, the information is used to handle expected CAN frames and to help in receive format processing.

**Table B.13 — CAN ID format (UNUM32) for ISO\_15765 and ISO\_11898**

Bit position	Name	Description
5,4	Padding Overwrite	00 = Use CP_CanFillerByteHandling for padding control 01 = Reserved 10 = Padding explicitly disabled for this CAN Id (overwrite CP_CanFillerByteHandling) 11 = Padding explicitly enabled for this CAN Id (overwrite CP_CanFillerByteHandling) NOTE These bits only apply to the CAN Ids transmitted by the MVCI, meaning that it only applies to the ComParams CP_CanPhysReqFormat and CP_CanFuncReqFormat
3	Addressing Scheme	0 = Normal addressing 1 = Extended addressing (Byte 1 of CAN Frame will contain N_AE/N_TA)
2	Data Transfer Handling	0 = UUDT Message (No PCI Bytes) 1 = USDT Message (Segmented frames)
1	CAN Id Size	0 = 11-bit 1 = 29-bit
0	Flow Control	0 = Flow Control frames are disabled 1 = Flow Control frames are enabled

Table B.14 provides a summary of all possible coded values for the CAN ID format ComParams.

**Table B.14 — Coded values for CAN ID format**

Coded value	Bit 5,4: Padding Overwrite	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description
0x00	normal	normal	unsegmented	11-bit	without FC	normal unsegmented 11-bit CP_CanFillerByteHandling for padding control
0x01	normal	normal	unsegmented	11-bit	with FC	n/a
0x02	normal	normal	unsegmented	29-bit	without FC	normal unsegmented 29-bit CP_CanFillerByteHandling for padding control
0x03	normal	normal	unsegmented	29-bit	with FC	n/a
0x04	normal	normal	segmented	11-bit	without FC	normal segmented 11-bit w/o FC CP_CanFillerByteHandling for padding control
0x05	normal	normal	segmented	11-bit	with FC	normal segmented 11-bit with FC CP_CanFillerByteHandling for padding control

Table B.14 — (continued)

Coded value	Bit 5,4: Padding Overwrite	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description
0x06	normal	normal	segmented	29-bit		normal segmented 29-bit w/o FC CP_CanFillerByteHandling for padding control
0x07	normal	normal	segmented	29-bit	with FC	normal segmented 29-bit with FC CP_CanFillerByteHandling for padding control
0x08	normal	extended	unsegmented	11-bit	without FC	extended unsegmented 11-bit CP_CanFillerByteHandling for padding control
0x09	normal	extended	unsegmented	11-bit	with FC	n/a
0x0A	normal	extended	unsegmented	29-bit	without FC	extended unsegmented 29-bit CP_CanFillerByteHandling for padding control
0x0B	normal	extended	unsegmented	29-bit	with FC	n/a
0x0C	normal	extended	segmented	11-bit	without FC	extended segmented 11-bit w/o FC CP_CanFillerByteHandling for padding control
0x0D	normal	extended	segmented	11-bit	with FC	extended segmented 11-bit with FC CP_CanFillerByteHandling for padding control
0x0E	normal	extended	segmented	29-bit	without FC	extended segmented 29-bit w/o FC CP_CanFillerByteHandling for padding control
0x0F	normal	extended	segmented	29-bit	with FC	extended segmented 29-bit with FC CP_CanFillerByteHandling for padding control
0x10-0x1F						Reserved
0x20	Disable padding	normal	unsegmented	11-bit	without FC	normal unsegmented 11-bit Do not use CP_CanFillerByteHandling for padding control
0x21	Disable padding	normal	unsegmented	11-bit	with FC	n/a
0x22	Disable padding	normal	unsegmented	29-bit	without FC	normal unsegmented 29-bit Do not use CP_CanFillerByteHandling for padding control
0x23	Disable padding	normal	unsegmented	29-bit	with FC	n/a
0x24	Disable padding	normal	segmented	11-bit	without FC	normal segmented 11-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x25	Disable padding	normal	segmented	11-bit	with FC	normal segmented 11-bit with FC Do not use CP_CanFillerByteHandling for padding control
0x26	Disable padding	normal	segmented	29-bit	without FC	normal segmented 29-bit w/o FC Do not use CP_CanFillerByteHandling for padding control

Table B.14 — (continued)

Coded value	Bit 5,4: Padding Overwrite	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description
0x27	Disable padding	normal	segmented	29-bit	with FC	normal segmented 29-bit with FC Do not use CP_CanFillerByteHandling for padding control
0x28	Disable padding	extended	unsegmented	11-bit	without FC	extended unsegmented 11-bit Do not use CP_CanFillerByteHandling for padding control
0x29	Disable padding	extended	unsegmented	11-bit	with FC	n/a
0x2A	Disable padding	extended	unsegmented	29-bit	without FC	extended unsegmented 29-bit Do not use CP_CanFillerByteHandling for padding control
0x2B	Disable padding	extended	unsegmented	29-bit	with FC	n/a
0x2C	Disable padding	extended	segmented	11-bit	without FC	extended segmented 11-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x2D	Disable padding	extended	segmented	11-bit	with FC	extended segmented 11-bit with FC Do not use CP_CanFillerByteHandling for padding control
0x2E	Disable padding	extended	segmented	29-bit	without FC	extended segmented 29-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x2F	Disable padding	extended	segmented	29-bit	with FC	extended segmented 29-bit with FC Do not use CP_CanFillerByteHandling for padding control
0x30	Enable padding	normal	unsegmented	11-bit	without FC	normal unsegmented 11-bit Do not use CP_CanFillerByteHandling for padding control
0x31	Enable padding	normal	unsegmented	11-bit	with FC	n/a
0x32	Enable padding	normal	unsegmented	29-bit	without FC	normal unsegmented 29-bit Do not use CP_CanFillerByteHandling for padding control
0x33	Enable padding	normal	unsegmented	29-bit	with FC	n/a
0x34	Enable padding	normal	segmented	11-bit	without FC	normal segmented 11-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x35	Enable padding	normal	segmented	11-bit	with FC	normal segmented 11-bit with FC Do not use CP_CanFillerByteHandling for padding control

Table B.14 — (continued)

Coded value	Bit 5,4: Padding Overwrite	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description
0x36	Enable padding	normal	segmented	29-bit	without FC	normal segmented 29-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x37	Enable padding	normal	segmented	29-bit	with FC	normal segmented 29-bit with FC Do not use CP_CanFillerByteHandling for padding control
0x38	Enable padding	extended	unsegmented	11-bit	without FC	extended unsegmented 11-bit Do not use CP_CanFillerByteHandling for padding control
0x39	Enable padding	extended	unsegmented	11-bit	with FC	n/a
0x3A	Enable padding	extended	unsegmented	29-bit	without FC	extended unsegmented 29-bit Do not use CP_CanFillerByteHandling for padding control
0x3B	Enable padding	extended	unsegmented	29-bit	with FC	n/a
0x3C	Enable padding	extended	segmented	11-bit	without FC	extended segmented 11-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x3D	Enable padding	extended	segmented	11-bit	with FC	extended segmented 11-bit with FC Do not use CP_CanFillerByteHandling for padding control
0x3E	Enable padding	extended	segmented	29-bit	without FC	extended segmented 29-bit w/o FC Do not use CP_CanFillerByteHandling for padding control
0x3F	Enable padding	extended	segmented	29-bit	with FC	extended segmented 29-bit with FC Do not use CP_CanFillerByteHandling for padding control

Table B.15 provides a list of the coded values for CP\_CanPhysReqFormat and CP\_CanFuncReqFormat CAN ID format ComParams. This list includes the applicable subset of coded values defined in Table B.14.

Table B.15 — Coded values for CP\_CanPhysReqFormat and CP\_CanFuncReqFormat

Coded value	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description for CP_CanPhysReqFormat and CP_CanFuncReqFormat
0x00	normal	unsegmented	11-bit	without FC	Normal addressing for Source (Tester), UUDT Message Transmit (No PCI Bytes), 11-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x01	normal	unsegmented	11-bit	with FC	n/a
0x02	normal	unsegmented	29-bit	without FC	Normal addressing for Source (Tester), UUDT Message Transmit (No PCI Bytes), 29-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x03	normal	unsegmented	29-bit	with FC	n/a

Table B.15 — (continued)

Coded value	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description for CP_CanPhysReqFormat and CP_CanFuncReqFormat
0x04	normal	segmented	11-bit	without FC	Normal addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 11-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x05	normal	segmented	11-bit	with FC	Normal addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 11-bit CAN ID Size, Flow Control frames are expected from Target (ECU)
0x06	normal	segmented	29-bit	without FC	Normal addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 29-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x07	normal	segmented	29-bit	with FC	Normal addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 29-bit CAN ID Size, Flow Control frames are expected from Target (ECU)
0x08	extended	unsegmented	11-bit	without FC	Extended addressing for Source (Tester), UUDT Message Transmit (No PCI Bytes), 11-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x09	extended	unsegmented	11-bit	with FC	n/a
0x0A	extended	unsegmented	29-bit	without FC	Extended addressing for Source (Tester), UUDT Message Transmit (No PCI Bytes), 29-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x0B	extended	unsegmented	29-bit	with FC	n/a
0x0C	extended	segmented	11-bit	without FC	Extended addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 11-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x0D	extended	segmented	11-bit	with FC	Extended addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 11-bit CAN ID Size, Flow Control frames are expected from Target (ECU)
0x0E	extended	segmented	29-bit	without FC	Extended addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 29-bit CAN ID Size, No Flow Control frames are expected from Target (ECU)
0x0F	extended	segmented	29-bit	with FC	Extended addressing for Source (Tester), USDT Message Transmit (Segmented transmits), 29-bit CAN ID Size, Flow Control frames are expected from Target (ECU)

NOTE Table B.15 does not explicitly consider the PaddingOverwrite bits (bits 4 and 5), but can be enhanced accordingly.

Table B.16 provides a list of the coded values for CP\_CanRespUSDTFormat CAN ID format ComParam. This list includes the applicable subset of coded values defined in Table B.14.

Table B.16 — Coded values for CP\_CanRespUSDTFormat

Coded value	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description for CP_CanRespUSDTFormat
0x0	normal	unsegmented	11-bit	without FC	n/a
0x1	normal	unsegmented	11-bit	with FC	n/a
0x2	normal	unsegmented	29-bit	without FC	n/a
0x3	normal	unsegmented	29-bit	with FC	n/a
0x4	normal	segmented	11-bit	without FC	Normal addressing for Target (ECU), USDT Message Receive (Segmented receives), 11-bit CAN ID Size, No Flow Control is sent when a First Frame is received
0x5	normal	segmented	11-bit	with FC	Normal addressing for Target (ECU), USDT Message Receive (Segmented receives), 11-bit CAN ID Size, Flow Control is sent when a First Frame is received
0x6	normal	segmented	29-bit	without FC	Normal addressing for Target (ECU), USDT Message Receive (Segmented receives), 29-bit CAN ID Size, No Flow Control is sent when a First Frame is received
0x7	normal	segmented	29-bit	with FC	Normal addressing for Target (ECU), USDT Message Receive (Segmented receives), 29-bit CAN ID Size, Flow Control is sent when a First Frame is received
0x8	extended	unsegmented	11-bit	without FC	n/a
0x9	extended	unsegmented	11-bit	with FC	n/a
0xA	extended	unsegmented	29-bit	without FC	n/a
0xB	extended	unsegmented	29-bit	with FC	n/a
0xC	extended	segmented	11-bit	without FC	Extended addressing for Target (ECU), USDT Message Receive (Segmented receives), 11-bit CAN ID Size, No Flow Control is sent when a First Frame is received
0xD	extended	segmented	11-bit	with FC	Extended addressing for Target (ECU), USDT Message Receive (Segmented receives), 11-bit CAN ID Size, Flow Control is sent when a First Frame is received
0xE	extended	segmented	29-bit	without FC	Extended addressing for Target (ECU), USDT Message Receive (Segmented receives), 29-bit CAN ID Size, No Flow Control is sent when a First Frame is received
0xF	extended	segmented	29-bit	with FC	Extended addressing for Target (ECU), USDT Message Receive (Segmented receives), 29-bit CAN ID Size, Flow Control is sent when a First Frame is received

NOTE The PaddingOverwrite bits (bits 4 and 5) do not apply to the ComParam CP\_CanRespUSDTFormat.

Table B.17 provides a list of the coded values for CP\_CanRespUUDTFormat CAN ID format ComParam. This list includes the applicable subset of coded values defined in Table B.14.

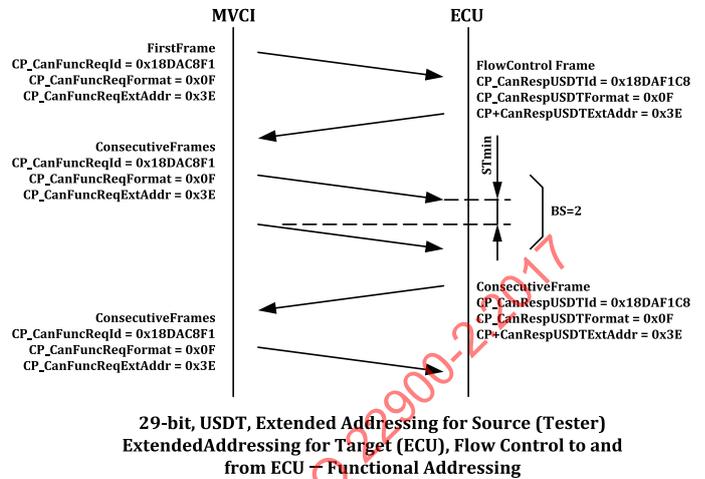
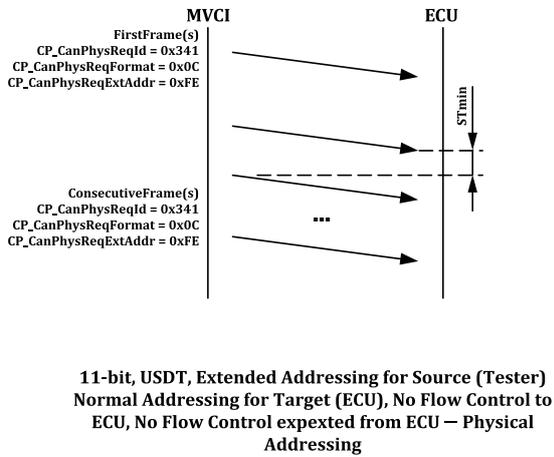
Table B.17 — Coded values for CP\_CanRespUUDTFormat

Coded value	Bit 3: Addressing Scheme	Bit 2: Data Transfer Handling	Bit 1: CAN ID Size	Bit 0: Flow Control	Description for CP_CanRespUUDTFormat
0x0	normal	unsegmented	11-bit	without FC	Normal addressing for Target (ECU), UUDT Message Receive (No PCI Bytes), 11-bit CAN ID Size, UUDT Message Receive (No Flow Control)
0x1	normal	unsegmented	11-bit	with FC	n/a
0x2	normal	unsegmented	29-bit	without FC	Normal addressing for Target (ECU), UUDT Message Receive (No PCI Bytes), 29-bit CAN ID Size, UUDT Message Receive (No Flow Control)
0x3	normal	unsegmented	29-bit	with FC	n/a
0x4	normal	segmented	11-bit	without FC	n/a
0x5	normal	segmented	11-bit	with FC	n/a
0x6	normal	segmented	29-bit	without FC	n/a
0x7	normal	segmented	29-bit	with FC	n/a
0x8	extended	unsegmented	11-bit	without FC	Extended addressing for Target (ECU), UUDT Message Receive (No PCI Bytes), 11-bit CAN ID Size, UUDT Message Receive (No Flow Control)
0x9	extended	unsegmented	11-bit	with FC	n/a
0xA	extended	unsegmented	29-bit	without FC	Extended addressing for Target (ECU), UUDT Message Receive (No PCI Bytes), 29-bit CAN ID Size, UUDT Message Receive (No Flow Control)
0xB	extended	unsegmented	29-bit	with FC	n/a
0xC	extended	segmented	11-bit	without FC	n/a
0xD	extended	segmented	11-bit	with FC	n/a
0xE	extended	segmented	29-bit	without FC	n/a
0xF	extended	segmented	29-bit	with FC	n/a

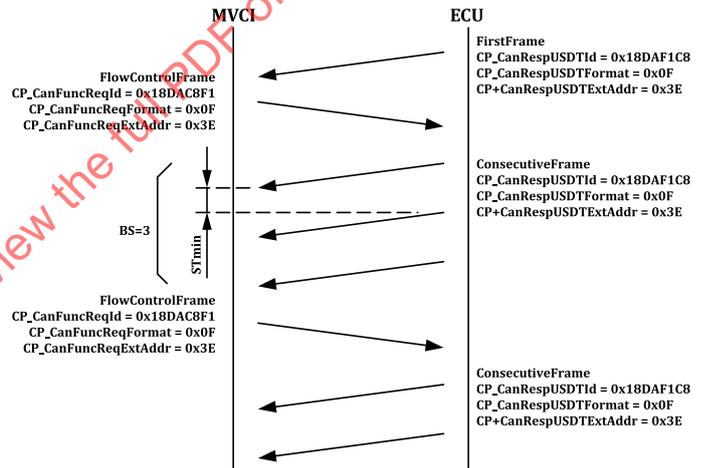
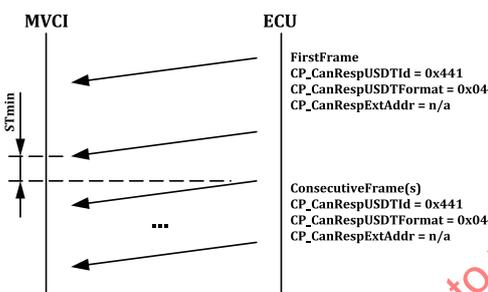
NOTE The PaddingOverwrite bits (bits 4 and 5) do not apply to the ComParam CP\_CanRespUUDTFormat.

B.4.4.1 CAN identifier format example

Multi-Frame Request



Multi-Frame Response



UUDT response

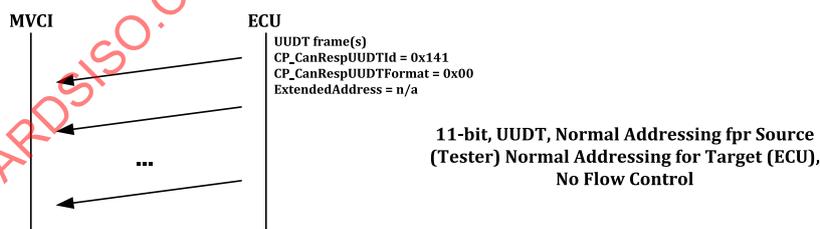


Figure B.2 — Example #1 CAN Id format configurations

B.4.4.2 CAN frame example

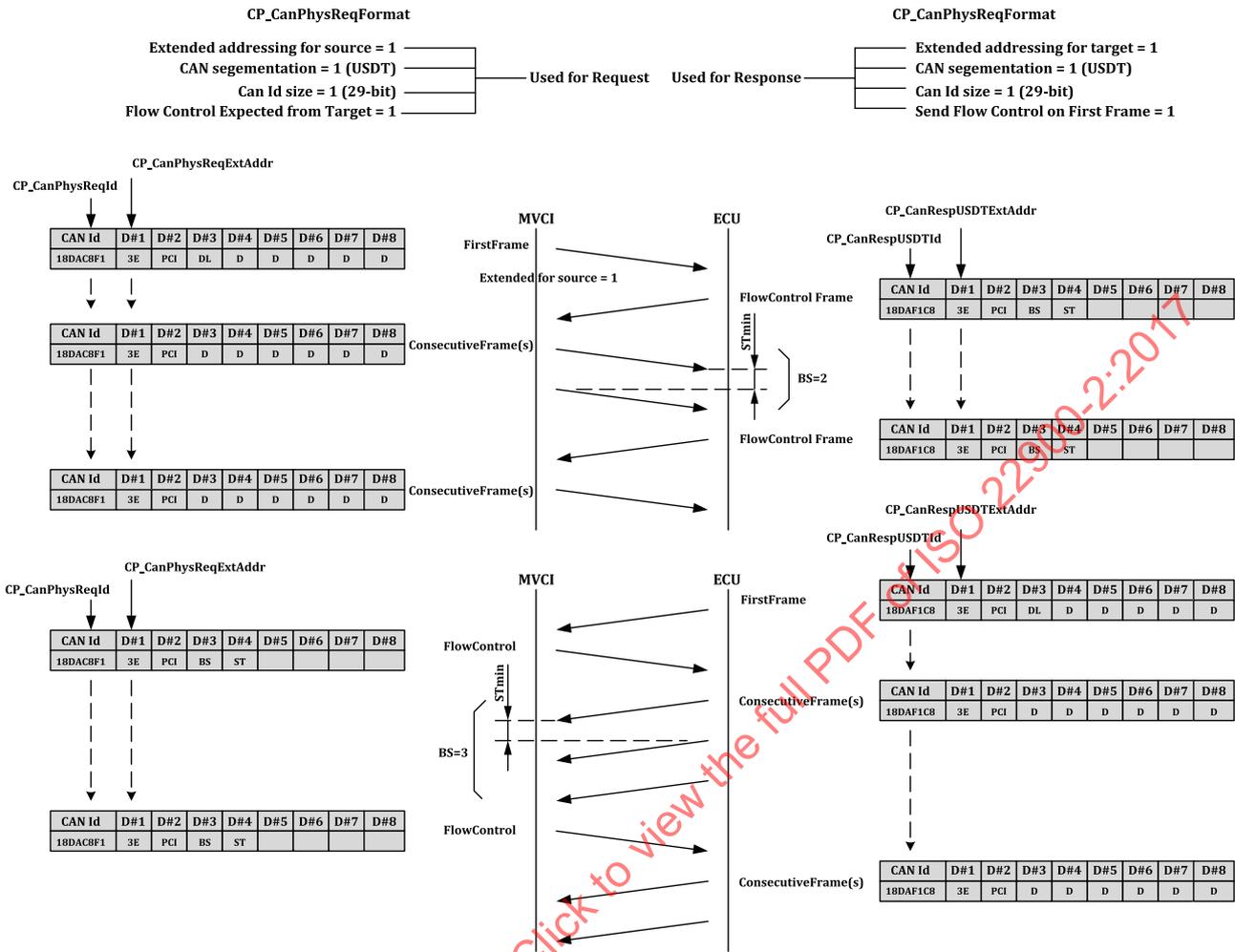


Figure B.3 — Example #2 CAN Id format configurations

### B.4.5 Non-CAN ComParam examples

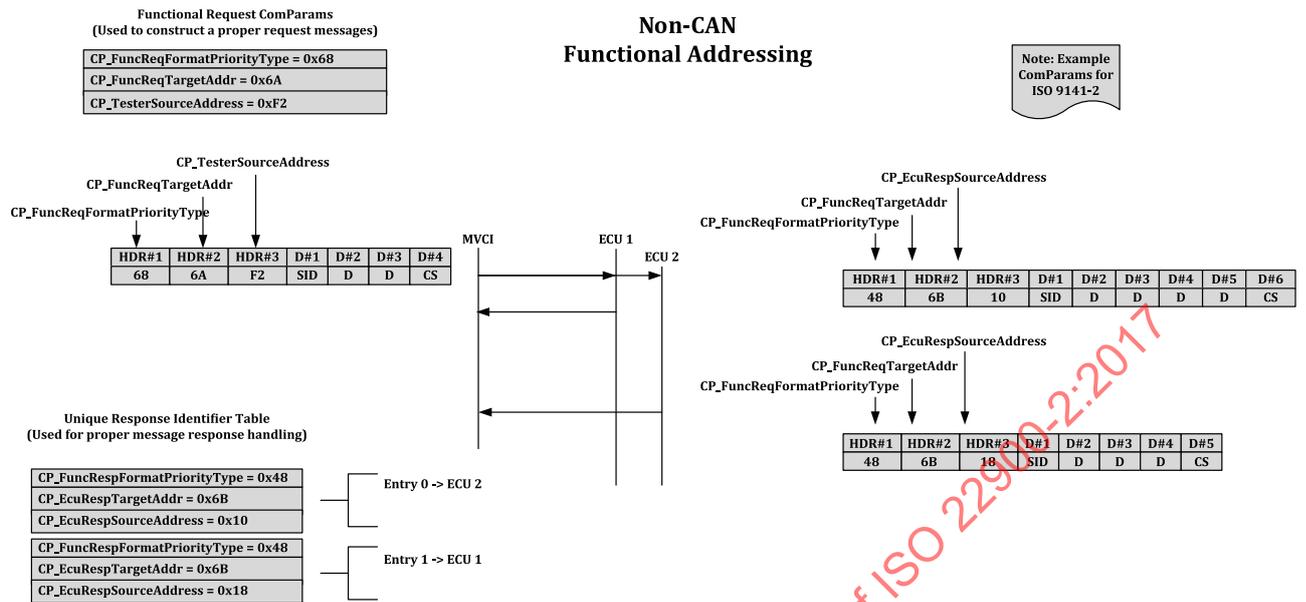


Figure B.4 — Example #1 Non-CAN functional addressing

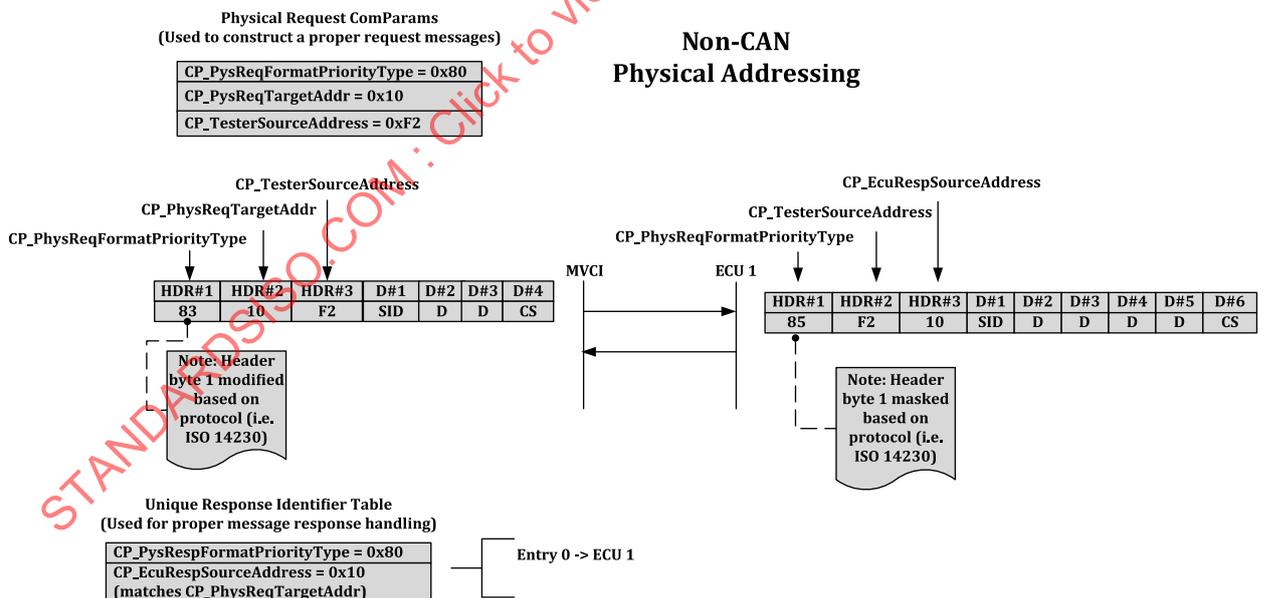


Figure B.5 — Example #2 Non-CAN physical addressing

**B.4.6 29-bit CAN identifier data page bits**

The Extended Data Page and Data Page bits determine which format of the 29 bit CAN identifier shall be used for the SAE J1939 and ISO 15765-3/ISO 14229-3 protocols.

**Table B.18 — Definition of Extended Data Page and Data Page field**

Extended Data Page Bit 25	Data Page Bit 24	Description
0	0	Page 0 PGNs SAE J1939 defined or manufacturer-defined “Normal Communication Message” strategy if SAE J1939 is not implemented
0	1	Page 1 PGNs SAE J1939 defined or manufacturer-defined “Normal Communication Message” strategy if SAE J1939 is not implemented
1	0	Reserved SAE J1939-reserved or manufacturer-defined “Normal Communication Message” strategy if SAE J1939 is not implemented
1	1	Used for ISO 15765-3/ISO 14229-3

**B.5 ComParam detailed descriptions**

**B.5.1 ComParam definitions for application layer**

**Table B.19 — Application layer ComParam definition table**

Short name	Detailed description	Default (by protocol)
CP_CanTransmissionTime	<p>Description: If the timeout values are used which have been received by the ECU via session control response (0x50), the Can transmission time has to be added to the timeout values. P2 = received P2 + CanTransmissionTime (contains delay for both transmission directions).</p> <p>Type: PDU_PT_UNUM32 Range: 0x0000-0xFFFFFFFF Resolution: 1 µs</p>	<p>ISO_15765_3 = 100000 ISO_14229_3 = 100000</p>
CP_ChangeSpeedCtrl	<p>Description: Control the behaviour of the MVCI protocol module in processing speed change messages. When this ComParam is enabled, the speed rate change will be activated on a successful Send or SendRecv ComPrimitive when the transmitted or received message matches the CP_ChangeSpeedMessage (baud rate as specified in CP_ChangeSpeedRate and termination resistor as specified in CP_ChangeSpeedResCtrl). In the case of monitoring mode, when a receive PDU is bound to a Receive Only ComPrimitive, and this ComParam is enabled, the speed rate change will also be activated and the corresponding ComParams will be interpreted accordingly. See use case example in Table I.1.</p>	All = 0 (disabled)

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = NO_SPDCHANGE – No speed change 1 = ENABLE_SPDCHANGE – Change Speed rate is enabled	
CP_ChangeSpeedMessage	Description: Switch Speed Message. The message is monitored for transmit and receive. When this message is detected on the vehicle bus, the CP_ChangeSpeedRate and CP_ChangeSpeedResCtrl ComParams are processed. NOTE 1 CP_ChangeSpeedCtrl is enabled for this ComParam to be active.	All Protocols: ParamActLen = 0 (not enabled)
	Type: PDU_PT_BYTEFIELD BYTEFIELD Format: ParamMaxLen = 12 ParamActLen = 0 to 12 pDataArray=ChangeSpdMessage[12] Range: Each byte = [0; 0xFF]	
CP_ChangeSpeedRate	Description: The data rate to be used when switching speed rates. When changed, this value is copied to CP_Baudrate ComParam.	All = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFFFFFFFF] Resolution: 1 bps	
CP_ChangeSpeedResCtrl	Description: This ComParam is used in conjunction with CP_ChangeSpeedCtrl. This ComParam is used to control automatic loading or unloading of the physical resource resistor when a change speed message has been transmitted or received.	All = 0 (DISABLE_AUTO_RESISTOR)
	Type: PDU_PT_UNUM32 Value: [0; 0xFF] 0=Not used (DISABLE_AUTO_RESISTOR) 1=AC load resistor (AUTO_LOAD_AC_RESISTOR) 2=60 Ohm load resistor (AUTO_LOAD_60OHM_RESISTOR) 3=120 Ohm load resistor (AUTO_LOAD_120OHM_RESISTOR) 4=SWCAN load resistor (AUTO_LOAD_SWCAN_RESISTOR) 0x80=Unload resistor (AUTO_UNLOAD_RESISTOR). NOTE 2 For AUTO_UNLOAD_RESISTOR, it is intended that CP_TerminationType be set to the initial value configured at the time of a PDUConnect.	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_ChangeSpeedTxDelay	Description: Minimum amount of time to wait before allowing the next transmit message on the Vehicle Bus after the successful transmission of a baud rate change message.	All = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFFFFFFFF] Resolution: 1 µs	
CP_CyclicRespTimeout	Description: This ComParam is used for ComPrimitives that have a NumRecvCycles set to IS-CYCLIC (-1, infinite) or to a value >0. The timer is enabled after the first positive response is received from an ECU. If CP_CyclicRespTimeout = 0, there is no receive timing enabled for the infinite receive ComPrimitive.	All protocols = 0
	Type: PDU_PT_UNUM32 Range: 0x0000-0xFFFFFFFF Resolution: 1 µs	
CP_DoIPConnectionCloseDelay	Description: Delay time before the Tester closes a TCP connection to a DoIP entity after the last logical link that uses this TCP connection has been disconnected via PDUDisconnect. NOTE 3 The Tester will close the TCP connection immediately, if PDUModuleDisconnect is called for a DoIP module using this TCP connection.	ISO_14229_5 = 0
	Type: PDU_PT_UNUM32 Range: 0x00000000 – 0xFFFFFFFF Resolution: 1 µs	
CP_EnablePerformanceTest	Description: This ComParam (when enabled) will place the tester into a performance measurement mode. Measurements will be collected during a normal ComPrimitive communications session. ComParams, such as P1Min, P2Min, Br and Cs, will be tested in this mode. Once the testing is disabled, results of the testing will be returned to the client application.	All protocols = 0
	Type: PDU_PT_UNUM32 Value [0;1] 0 = Disabled 1 = Enabled	
CP_EnableInitSeqRepetition	Description: This ComParam allows to enable repetition of fast-init and 5-baud initialization sequences when a request is repeated caused by CP_RepeatReqCountApp. If enabled, fast-init and 5-baud initialization sequences will also be repeated.	ISO_14230_4=0 ISO_9141_2=0 ISO_14230_3=0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Disabled (default) 1 = Enabled	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_Loopback	<p>Description: Echo Transmitted messages in the receive queue, including periodic messages. Loopback messages shall only be sent after successful transmission of a message. Loopback frames are not subject to message filtering.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 1] 0 = OFF 1 = ON</p>	All Protocols=0
CP_MessageIndicationRate	<p>Description: The maximum rate for which the D-PDU API will generate PDU_EVENT_ITEM of PDU_IT_RESULT type for a receive only ComPrimitive. The D-PDU API will generate PDU_IT_RESULT events at time intervals greater than or equal to the value defined by this ComParam even if the ECU is sending the data at a faster rate. The function of this parameter is to prevent a broadcast PGN from an ECU that is sent at a high frequency from flooding the client with data.</p> <p>Type: PDU_PT_UNUM32 Range: 0x00000000 disabled 0x00000001 - 0xFFFFFFFF Resolution: 1 µs</p>	SAE_J1939_73=0
CP_ModifyTiming	<p>Description: This parameter signals the D-PDU API to automatically modify timing parameters based on a response from the ECU. For ISO 14230-2, this would apply to service 0x83/0xC3 with TPI of 1, 2 or 3. For ISO 15765-3/ISO 14229-3, this would apply to service 0x10/0x50. For functional addressing mode, the worst case timing parameter returned by the responding ECUs shall be used. Based on the protocol, the following parameters are modified when a positive ECU response is received:</p> <ul style="list-style-type: none"> <li>— CP_P2<sub>Max</sub></li> <li>— CP_P2<sub>Min</sub></li> <li>— CP_P2<sub>Star</sub></li> <li>— CP_P3<sub>Min</sub></li> <li>— CP_P4<sub>Min</sub></li> </ul> <p>NOTE 4 The values returned by an ECU are in a different time resolution than the ComParams to be automatically modified. The values will be reinterpreted from the protocol specified time resolution to the 1 µs resolution specified in the D-PDU API.</p> <p>Type: PDU_PT_UNUM32 Value: [0;1] 0 = Disabled 1 = Enabled</p>	ISO_15765_3 = 0 ISO_14229_3 = 0 ISO_14230_4 = 0 ISO_14230_3 = 0 ISO_14229_5 = 0

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_NetworkTransmissionTime	<p>Description: If the timeout values are used which have been received by the ECU via session control response (0x50), the Network transmission time has to be added to the timeout values.</p> <p><math>P2 = \text{received } P2 + \text{NetworkTransmissionTime}</math> (contains delay for both transmission directions).</p> <p>NOTE 5 This ComParam is relevant for all protocols except those on CAN (especially Ethernet). For CAN protocols, the ComParam CP_CanTransmissionTime is relevant instead.</p>	ISO_14229_5 = 100000
	<p>Type: PDU_PT_UNUM32</p> <p>Range: 0x00000000 – 0xFFFFFFFF</p> <p>Resolution: 1 <math>\mu</math>s</p>	
CP_P2Max	<p>Description: Timeout in receiving an expected frame after a successful transmit complete. Also used for multiple ECU responses.</p>	<p>ISO_15765_4=50000</p> <p>ISO_14230_4=50000</p> <p>ISO_9141_2 =50000</p> <p>SAE_J1850_VPW=100000</p> <p>SAE_J1850_PWM=100000</p> <p>ISO_14230_3=50000</p> <p>ISO_15765_3=150000</p> <p>ISO_14229_3=150000</p> <p>ISO_11898_RAW = 50000</p> <p>SAE_J1587= 60000000</p> <p>SAE_J1939_73 = 200000</p> <p>SAE_J2190=50000</p>
	<p>Type: PDU_PT_UNUM32</p> <p>Range: [0; 125000000]</p> <p>Resolution: 1 <math>\mu</math>s</p>	
CP_P2Max_Ecu	<p>Description: Performance requirement for the server to start with the response message after the reception of a request message (indicated via N_USData.ind). This is a performance requirement ComParam.</p> <p><math>(CP\_P2_{\text{Max\_Ecu}} &lt; CP\_P2_{\text{Max}} - CP\_CanTransmissionTime)</math></p> <p>NOTE 6 <math>CP\_P2_{\text{Max\_Ecu}} &lt; CP\_P2_{\text{Max}} - CP\_CanTransmissionTime</math>.</p> <p>NOTE 7 For ISO_14229_5, this is the complete time, including the network transmission time.</p>	<p>ISO_15765_4 = 40000</p> <p>ISO_14230_4 = 50000</p> <p>ISO_9141_2 = 50000</p> <p>ISO_14230_3 = 50000</p> <p>ISO_15765_3 = 50000</p> <p>ISO_14229_3 = 50000</p>
	<p>Type: PDU_PT_UNUM32</p> <p>Range: [0; 100000000]</p> <p>Resolution: 1 <math>\mu</math>s</p>	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_P2Min	Description: This sets the minimum time between tester request and ECU responses or two ECU responses. After the request, the interface shall be capable of handling an immediate response (P2_min=0). For subsequent responses, a byte received after P1_MAX shall be considered as the start of the subsequent response. This is a performance requirement ComParam.	ISO_15765_4=0 ISO_14230_4=25000 ISO_9141_2 =25000 SAE_J1850_VPW = 0 SAE_J1850_PWM = 0 ISO_14230_3 = 25000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	ISO_15765_3 = 0 ISO_14229_3 = 0 ISO_11898 = 0 SAE_J2190 = 0
CP_P2Star	Description: Timeout for the client to expect the start of the response message after the reception of a negative response message (indicated via N_USData.ind) with response code 0x78 (enhanced response timing). See CP_RC78Handling for details describing 0x78 0x7F handling. This parameter is used for all protocols that support the negative response code 0x78. For some protocols it is used instead of the recommended P3Max parameter.	ISO_15765_3=5050000 ISO_14229_3=5050000 ISO_15765_4=5000000 (see ISO 15765-3/ ISO_14229-3 for delta P2) ISO_J1850_VPW = 5000000 ISO_J1850_PWM = 5000000
	Type: PDU_PT_UNUM32 Range: [0; 655350000] Resolution: 1 µs	ISO_14230_3 = 5000000 ISO_14230_4 = 5000000 SAE_J2190=5000000
CP_P2Star_Ecu	Description: Performance requirement for the server to start with the response message after the transmission of a negative response message (indicated via N_USData.con) with response code 0x78 (enhanced response timing). This is a performance requirement ComParam. NOTE 8 CP_P2Star_Ecu < CP_P2Star - 0.5 * CP_CanTransmissionTime	ISO_15765_4=5000000 ISO_15765_3=5000000 ISO_14229_3=5000000
	Type: PDU_PT_UNUM32 Range: [0; 655350000] Resolution: 1 µs	
CP_P3Func	Description: Minimum time for the client to wait after the successful transmission of a functionally addressed request message (indicated via N_USData.con) before it can transmit the next functionally addressed request message in case no response is required or the requested data is only supported by a subset of the functionally addressed servers.	ISO_15765_3 = 150000 ISO_14229_3 = 150000 ISO_11898_RAW = 0 ISO_14229_5 = 50000
	Type: PDU_PT_UNUM32 Range: [0; 125000000] Resolution: 1 µs	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_P3Max_Ecu	Description: Time between end of ECU responses and start of new tester request.	ISO_14230_4=5000000 ISO_14230_3=5000000
	Type: PDU_PT_UNUM32 Range: [0; 100000000] Resolution: 1 µs	ISO_9141_2 = 5000000
CP_P3Min	Description: Minimum time between end of non-negative ECU responses and start of new request. The interface will accept all responses up to CP_P3Min time. The interface will allow transmission of a request any time after CP_P3Min. (See CP_RC21RequestTime for minimum time between end of ECU negative responses and start of new requests.)	ISO_9141_2=55000 ISO_14230_4 = 55000 ISO_14230_3 = 55000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	
CP_P3Phys	Description: Minimum time for the client to wait after the successful transmission of a physically addressed request message (indicated via N_USData.con) with no response required before it can transmit the next physically addressed request message.	ISO_15765_3 = 150000 ISO_14229_3 = 150000 ISO_11898_RAW = 0 ISO_14229_5 = 50000
	Type: PDU_PT_UNUM32 Range: [0; 125000000] Resolution: 1 µs	
CP_P6Max	Description: Timeout for the client to wait after the successful transmission of a request message (indicated via T_Data.con) for the complete reception of the corresponding response message (indicated via T_Data.ind). If the transport/network layer does not support the T_DataSOM.ind interface (e.g. ISO 13400 DoIP), this ComParam shall be used instead of CP_P2Max.	ISO_14229_5 = 1000000
	Type: PDU_PT_UNUM32 Range: [0; 125000000] Resolution: 1 µs Class: TIMING Support, Usage: S,T	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_P6Star	Description: Enhanced timeout for the client to wait after the reception of a negative response message with negative response code 0x78 (indicated via T_Data.ind) for the complete reception of the corresponding response messages (indicated via T_Data.ind). If the transport/network layer doesn't support the T_DataSOM.ind interface (e.g. ISO 13400 DoIP), this ComParam shall be used instead of CP_P2Star.	ISO_14229_5 = 1000000
	Type: PDU_PT_UNUM32 Range: [0; 655350000] Resolution: 1 µs Class: TIMING Support, Usage: S,T	
CP_RC21CompletionTimeout	Description: Time period the tester accepts repeated negative responses with response code 0x21 and repeats the same request. Timer is started after reception of first negative response.	ISO_15765_4 = 1300000 ISO_14230_4=1300000 SAE_J1850_VPW=0 SAE_J1850_PWM=0 ISO_14230_3=1300000 ISO_15765_3 = 1300000 ISO_14229_3 = 1300000 SAE_J2190=0 ISO_14229_5 = 1300000
	Type: PDU_PT_UNUM32 Range: [0; max] (max is protocol specific) Resolution: 1 µs	
CP_RC21Handling	Description: Repetition mode in case of response code 0x7F XX 0x21.	ISO_15765_4=2 ISO_14230_4=2 SAE_J1850_VPW=0 SAE_J1850_PWM=0 ISO_14230_3=0 ISO_15765_3=0 ISO_14229_3=0 SAE_J2190=0 ISO_14229_5 = 0
	Type: PDU_PT_UNUM32 Value: [0; 2] 0 = Disabled 1 = Continue handling negative responses until CP_RC21CompletionTimeout 2 = Continue handling unlimited (until disabled)	
CP_RC21RequestTime	Description: Time between negative response with response code 0x21 and the retransmission of the same request. If CP_P3Min is greater than CP_RC21RequestTime, the time delay prior to the retransmission of the same request will be CP_P3Min.	ISO_15765_4= 200000 ISO_14230_4=200000 SAE_J1850_VPW=200000 SAE_J1850_PWM=200000 ISO_14230_3=200000 ISO_15765_3=10000 ISO_14229_3=10000 SAE_J2190 =200000 ISO_14229_5 = 200000
	Type: PDU_PT_UNUM32 Range: [0; 100000000] Resolution: 1 µs	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_RC23CompletionTimeout	Description: Time period the tester accepts repeated negative responses with response code 0x23 and repeats the same request.	ISO_15765_4=0 SAE_J1850_VPW=0 SAE_J1850_PWM=0
	Type: PDU_PT_UNUM32 Range: [0; max] (max is protocol specific) Resolution: 1 µs	ISO_14230_3=0 ISO_15765_3=0 ISO_14229_3=0 SAE_J2190=0 ISO_14229_5 = 0
CP_RC23Handling	Description: Repetition mode in case of response code 0x7F XX 0x23.	ISO_15765_4=0 SAE_J1850_VPW=0 SAE_J1850_PWM=0
	Type: PDU_PT_UNUM32 Value: [0; 2] 0 = Disabled 1 = Continue handling negative responses until CP_RC23CompletionTimeout 2 = Continue handling unlimited (until disabled)	ISO_14230_3=0 ISO_15765_3=0 ISO_14229_3=0 SAE_J2190=0 ISO_14229_5 = 0
CP_RC23RequestTime	Description: The time the D-PDU API waits to re-request the message when receiving a negative response code 0x23. For some protocols (SAE_J1850_VPW), it is possible to get a positive response after receiving a negative response code 0x23, so the D-PDU API uses this ComParam as the time to receive a possible positive response before making the re-request. On a positive response within this time, the re-request is cancelled. The D-PDU API postpones the re-request until the timeout of CP_RC23RequestTime (or a CP_P3Min timeout, in case CP_P3Min is greater than CP_RC23RequestTime). For ISO 14230-3, there will be no positive response following a RC23; therefore, the D-PDU API is expected to always make a re-request if enabled (CP_RC23Handling != 0). The cycle of receiving negative response code 0x23 and retransmitting the request continues until CP_RC23CompletionTimeout expires (applicable only if CP_RC23Handling is set to 1).	ISO_15765_4=200000 SAE_J1850_VPW=200000 SAE_J1850_PWM=200000 ISO_14230_3=200000 ISO_15765_3=0 ISO_14229_3=0 SAE_J2190=200000 ISO_14229_5 = 200000
	Type: PDU_PT_UNUM32 Range: [0; 100000000] Resolution: 1 µs	

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_RC78CompletionTimeout	<p>Description: Time period the tester accepts repeated negative responses with response code 0x78 and waits for a positive response further on.</p> <p>Type: PDU_PT_UNUM32 Range: [0; max] (max is protocol specific) Resolution: 1 µs</p>	ISO_15765_4=30000000 ISO_14230_4=30000000 SAE_J1850_VPW=25000000 SAE_J1850_PWM=25000000 ISO_14230_3=25000000 ISO_15765_3=25000000 ISO_14229_3=25000000 SAE_J2190=25000000 ISO_14229_5 = 25000000
CP_RC78Handling	<p>Description: Handling of 0x7F XX 0x78 ResponseTimeout and 0x78 Repetitions. The receive timeout value will be CP_P2Star. This timer will be reset on each consecutive reception of the 0x7F 0x78 response.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 2] 0 = Disabled 1 = Continue handling negative responses until CP_RC78CompletionTimeout 2 = Continue handling unlimited (until disabled)</p>	ISO_15765_4=2 ISO_14230_4=2 SAE_J1850_VPW=0 SAE_J1850_PWM=0 ISO_14230_3=2 ISO_15765_3=2 ISO_14229_3=2 SAE_J2190=0 ISO_14229_5 = 2
CP_RCByteOffset	<p>Description: This parameter is used by the MSCI Protocol Handlers to offset into the received negative response message (0x7F) to retrieve the response code byte. Most protocols as a default place the response code as the last byte of the message. There are some protocols which place the response code after the Service Id (offset = 1). A range is provided to allow for different negative response configurations.</p> <p>Type: PDU_PT_UNUM32 Value: [1;0xFFFFFFFF]: 0 = invalid value 1 = first byte after the Service Id byte (0x7F) 0xFFFFFFFF = last byte in message (not including checksum bytes)</p>	All protocols = 0xFFFFFFFF (last byte in message)
CP_RepeatReqCountApp	<p>Description: This ComParam contains a counter to enable a re-transmission of the last request when either a transmit, receive error or timeout with no response is detected. This only applies to the application layer.</p> <p>This ComParam applies to re-trying of fast-init and 5-baud initialization sequences based on value of ComParam CP_EnableInitSeqRepetition.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 127500] Resolution: 1 count</p>	All protocols = 0

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_SessionTiming_Ecu	<p>Description: Timing parameters to be used by different sessions for the ECU protocol application layer ISO15765_3/ISO14229_3 in response to a service 0x10 (Set Diagnostic Session). The ECU returns the session timing information on a positive response to the service. If CP_ModifyTiming is enabled, the MVCI protocol module will interpret the positive response from the ECU and set the appropriate timing ComParams.</p> <p>Type: PDU_PT_STRUCTFIELD STRUCTFIELD Format: ComParamStructType= PDU_CPST_SESSION_TIMING ParamMaxEntries = 255 ParamActEntries = 0 to 255 pStructArray=PDU_PARAM_STRUCT_SESS_TIMING</p>	<p>ISO_15765_3: ParamActEntries = 0 (not enabled) ISO_14229_3: ParamActEntries = 0 (not enabled)</p>
CP_SessionTimingOverride	<p>Description: This parameter signals the D-PDU API to override the response from any ECUs to a Session Timing request (see CP_SessionTiming_Ecu and CP_ModifyTiming). The timing parameters are to be used for the ECU protocol application layer ISO15765_3/ISO14229_3 in response to a service 0x10 (Set Diagnostic Session). The ECU returns the session timing information on a positive response to the service. If CP_ModifyTiming is enabled and CP_SessionTimingOverride is not empty (ParamActEntries != 0), then the MVCI protocol module will use data in this ComParam instead of the data returned in a positive response from the ECUs.</p> <p>Type: PDU_PT_STRUCTFIELD STRUCTFIELD Format: ComParamStructType= PDU_CPST_SESSION_TIMING ParamMaxEntries = 255 ParamActEntries = 0 to 255 pStructArray=PDU_PARAM_STRUCT_SESS_TIMING</p>	<p>ISO_15765_3: ParamActEntries = 0 (not enabled) ISO_14229_3: ParamActEntries = 0 (not enabled)</p>
CP_StartMsgIndEnable	<p>Description: Start Message Indication Enable. Upon receiving a first frame of a multi-frame message (ISO 15765) or upon receiving a first byte of a UART message, an indication will be set in the RX result item. No data bytes will accompany the result item.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Start Message Indication Disabled 1 = Start Message Indication Enabled</p>	<p>ISO_15765_4=0 ISO_15765_3=0 ISO_14229_3=0 ISO_14230_4=0 ISO_14230_3=0 ISO_9141_2=0 SAE_J1939=0 SAE_J1708=0</p>

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_SuspendQueueOnError	<p>Description: This ComParam is used as a temporary ComParam for services that require a positive response before any further Com Primitives can be executed.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Do not suspend ComPrimitive Transmit Queue 1 = Suspend ComPrimitive Transmit Queue on a Timeout Error or on a non-handled 0x7F error (not an enabled protocol ComParam)</p>	ISO_15765_4=0 ISO_14230_4=0 ISO_9141_2=0 ISO_14230_3=0 ISO_15765_3=0 ISO_14229_3=0 SAE_J1850_VPW=0 SAE_J1850_PWM=0 SAE_J2190=0 SAE_J1939_73=0
CP_SwCan_HighVoltage	<p>Description: Indicates that the Single Wire CAN message should be transmitted as a High-Voltage Message. Simultaneously transmitting in high voltage and high speed mode will result in undefined behaviour. This ComParam is only applicable when the Bus Type selected is SAE_J2411_SWCAN.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Normal Message 1 = High-Voltage Message</p>	ISO_15765_3=0 ISO_14229_3=0 ISO_14230_3=0 ISO_11898_RAW=0
CP_TesterPresentAddrMode	<p>Description: Addressing Mode to be used for periodic Tester Present messages.</p> <p>Uses the PhysReqxxx or FuncReqxxx ComParams.</p> <p>NOTE 9 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 10 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Use Physical Addressing for the Tester Present message. 1 = Use Functional Addressing for the Tester Present message.</p>	ISO_15765_4=1 ISO_14230_4=1 ISO_9141_2=1 SAE_J1850_VPW=0 SAE_J1850_PWM=0 ISO_14230_3=0 ISO_15765_3=1 ISO_14229_3=1 ISO_14229_5 = 1

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_TesterPresentExpPosResp	<p>Description: Define the expected ECU positive response to a Tester Present Message. This is only applicable if CP_TesterPresentReqRsp is set to 1 (ECU responses are expected on a Tester Present Message). No header bytes or checksum bytes are included. Only the ParamActLen bytes in the array will be compared with the received ECU data.</p> <p>NOTE 11 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 12 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <p>Type: PDU_PT_BYTEFIELD            BYTEFIELD Format:            ParamMaxLen = 12            ParamActLen = 0 to 12            pDataArray=TesterPresentExpPosResp[12]            Range: Each byte = [0; 0xFF]</p>	<p>ISO_15765_4: ParamActLen = 2, pDataArray = {0x41, 0x00}</p> <p>ISO_14230_4: ParamActLen = 2, pDataArray = {0x41, 0x00}</p> <p>ISO_9141_2: ParamActLen = 2, pDataArray = {0x41, 0x00}</p> <p>SAE_J1850_VPW:            ParamActLen = 0            (not enabled)</p> <p>SAE_J1850_PWM:            ParamActLen = 0            (not enabled)</p> <p>ISO_14230_3: ParamActLen = 1, pDataArray = {0x7E}</p> <p>ISO_15765_3: ParamActLen = 0 (not enabled)</p> <p>ISO_14229_3: ParamActLen = 0 (not enabled)</p> <p>ISO_13400_2:            ParamActLen=0            (not enabled)</p>

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_TesterPresentExpNegResp	<p>Description: Define the expected ECU negative response to a Tester Present Message. This is only applicable if CP_TesterPresentReqRsp is set to 1 (ECU responses are expected on a Tester Present Message). No header bytes or checksum bytes are included.</p> <p>When a negative response is received to a tester present message, which cannot be handled by the MSCI Protocol module (see RC 21, RC 23 and RC 78), the MSCI protocol module should report a Tester Present Error, but continue sending Tester Present Messages.</p> <p>(See PDU_ERR_EVT_TESTER_PRESENT_ERROR)</p> <p>NOTE 13 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 14 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <p>Type: PDU_PT_BYTEFIELD            BYTEFIELD Format:            ParamMaxLen = 12            ParamActLen = 0 to 12            pDataArray=TesterPresentExpNegResp[12]            Range: Each byte = [0; 0xFF]</p>	ISO_15765_4: ParamActLen = 1, pDataArray = {0x7F} ISO_14230_4: ParamActLen = 1, pDataArray = {0x7F} ISO_9141_2: ParamActLen = 1, pDataArray = {0x7F} SAE_J1850_VPW: ParamActLen = 0 (not enabled) SAE_J1850_PWM: ParamActLen = 0 (not enabled) ISO_14230_3: ParamActLen = 2, pDataArray = {0x7F, 0x3E} ISO_15765_3: ParamActLen = 0 (not enabled) ISO_14229_3: ParamActLen = 0 (not enabled) ISO_14229_5: ParamActLen=0 (not enabled)
CP_TesterPresentHandling	<p>Description: Define Tester Present message generation settings. The ComLogicalLink shall be in the state PDU_CLLST_COMM_STARTED to enable tester present message handling. (See PDU_COPT_STARTCOMM ComPrimitive.)</p> <p>NOTE 15 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled, any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 16 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <p>Type: PDU_PT_UNUM32            Value: [0; 1]            0 = Do not generate Tester Present messages            1 = Generate Tester Present messages</p>	ISO_15765_4=0 ISO_14230_4=1 ISO_9141_2=1 SAE_J1850_VPW=0 SAE_J1850_PWM=0 ISO_14230_3=1 ISO_15765_3=1 ISO_14229_3=1 ISO_14229_5 = 1

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_TesterPresentMessage	<p>Description: Define the Tester Present Message. This message data does not include any header bytes or checksum information.</p> <p>NOTE 17 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 18 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <hr/> <p>Type: PDU_PT_BYTEFIELD                      BYTEFIELD Format:                      ParamMaxLen = 12                      ParamActLen = 0 to 12                      pDataArray=TesterPresentMessage[12]                      Range: Each byte = [0; 0xFF]</p>	<p>ISO_15765_4: ParamActLen = 2, pDataArray = {0x01, 0x00}</p> <p>ISO_14230_4: ParamActLen = 2, pDataArray = {0x01, 0x00}</p> <p>ISO_9141_2: ParamActLen = 2, pDataArray = {0x01, 0x00}</p> <p>SAE_J1850_VPW: ParamActLen = 0 (not enabled)</p> <p>SAE_J1850_PWM: ParamActLen = 0 (not enabled)</p> <p>ISO_14230_3: ParamActLen = 1, pDataArray = {0x3E}</p> <p>ISO_15765_3: ParamActLen = 2, pDataArray = {0x3E, 0x80}</p> <p>ISO_14229_3: ParamActLen = 2, pDataArray = {0x3E, 0x80}</p> <p>ISO_14229_5: ParamActLen = 2, pDataArray = {0x3E, 0x80}</p>
CP_TesterPresentReqRsp	<p>Description: Define settings for handling Tester Present ECU responses.</p> <p>NOTE 19 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 20 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <hr/> <p>Type: PDU_PT_UNUM32                      Value: [0; 1]                      0=No response returned by an ECU on a Tester Present message.                      1=An ECU response is expected from a Tester Present message. The response message will be discarded by the MVCI protocol module. See ComParams (CP_TesterPresentExpPosResp and CP_TesterPresentExpNegResp) for proper response handling.</p>	<p>ISO_15765_4=0</p> <p>ISO_14230_4=1</p> <p>ISO_9141_2=1</p> <p>SAE_J1850_VPW=0</p> <p>SAE_J1850_PWM=0</p> <p>ISO_14230_3=1</p> <p>ISO_15765_3=0</p> <p>ISO_14229_3=0</p> <p>ISO_14229_5 = 0</p>

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_TesterPresentSendType	<p>Description: Define settings for the type of tester present transmits.</p> <p>NOTE 21 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 22 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Send on periodic interval defined by CP_TesterPresentTime 1 = Send when bus has been idle for CP_TesterPresentTime</p>	ISO_15765_4=0 ISO_14230_4=1 ISO_9141_2=1 SAE_J1850_VPW=0 SAE_J1850_PWM=0 ISO_14230_3=1 ISO_15765_3=0 ISO_14229_3=0 ISO14229-5 = 0
CP_TesterPresentTime	<p>Description: Time between Tester Present messages or Time bus shall be idle before transmitting a Tester Present Message.</p> <p>NOTE 23 If the CLL is in the PDU_CLLST_COMM_STARTED state and tester present handling is enabled (see CP_TesterPresentHandling), any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.</p> <p>NOTE 24 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP_P3Min, CP_P3Func and CP_P3Phys.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 30000000] Resolution: 1 µs</p>	ISO_15765_4=2000000 ISO_14230_4=2000000 SAE_J1850_VPW=2000000 SAE_J1850_PWM=2000000 ISO_14230_3=2000000 ISO_15765_3=2000000 ISO_15765_3=2000000 ISO_14229_3=2000000 SAE_J2190=2000000 ISO_14229_5 = 2000000
CP_TesterPresentTime_Ecu	<p>Description: Time for the server to keep a diagnostic session (other than the default session) active while not receiving any diagnostic request message.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 30000000] Resolution: 1 µs</p>	ISO_15765_4=5000000 ISO_14230_4=5000000 SAE_J1850_VPW=5000000 SAE_J1850_PWM=5000000 ISO_14230_3=5000000 ISO_15765_3=5000000 ISO_14229_3=5000000 SAE_J2190=5000000

Table B.19 — (continued)

Short name	Detailed description	Default (by protocol)
CP_TransmitIndEnable	Description: Transmit Indication Enable. On completion of a transmit message by the protocol, an indication will be set in the RX_FLAG result item. No data bytes will accompany the result item. NOTE 25 For DoIP, Transmit Indication is triggered by DoIP Acknowledge.	All Protocols = 0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Transmit Indication Disabled 1 = Transmit Indication Enabled	

## B.5.2 ComParam definitions for transport layer

Table B.20 — Transport layer detailed ComParam table

Short name	Detailed description	Default (by protocol)
CP_5BaudAddressInverted	Description: Logically inverted value of 5 baud address returned by the ECU during binitialization.	ISO_9141_2 = 0xCC ISO_14230_2 = 0xCC
	Type: PDU_PT_UNUM32 Value: [0; 0xFF]	
CP_5BaudAddressFunc	Description: Value of 5 baud address in case of functional-addressed communication. The correct baud rate address type (functional/physical) is selected during execution of a Start Communication ComPrimitive based on the setting of the CP_RequestAddrMode ComParam.	ISO_9141_2=0x33 ISO_14230_2=0x33 ISO_14230_4=0x33 Ecu Variant Specific
	Type: PDU_PT_UNUM32 Value: [0; 0xFF]	
CP_5BaudAddressPhys	Description: Value of 5 baud address in case of physical-addressed communication. The correct baud rate address type (functional/physical) is selected during execution of a Start Communication ComPrimitive based on the setting of the CP_RequestAddrMode ComParam.	ISO_9141_2=0x01 ISO_14230_2=0x01 Ecu Variant Specific
	Type: PDU_PT_UNUM32 Value: [0; 0xFF]	
	Type: PDU_PT_UNUM32 Value[0;3]: 0 = Init as defined in ISO 9141-2 and ISO 14230-4 1 = ISO 9141 init followed by interface sending inverted key byte 2, no inverted address 2 = ISO 9141 init followed by ECU sending inverted address, no inverted key byte 2 3 = Init as defined in ISO 9141, no inverted key byte 2 nor inverted address	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_5BaudMode	Description: Type of 5-baud initialization. This ComParam allows either ISO 9141 initialization sequence, ISO 9141-2/ISO 14230 initialization sequence or hybrid versions, which include only one of the extra bytes defined for ISO 9141-2 and ISO 14230.  (Initialization for ISO 9141-2 and ISO 14230 include the init sequence as defined in ISO 9141 plus inverted key byte 2 sent from the interface to the ECU and the inverted address sent from the ECU to the interface.)	ISO_9141_2=0 ISO_14230_2=0
CP_5BaudCommBaudrateOverride	Description: Used to specify if the baudrate calculated based on the 5 baud init sync pattern received from the ECU shall be used (default) or the value given in the ComParam CP_Baudrate.  The resulting baudrate applies for any communication that takes place following the sync byte (either the detected one or CP_Baudrate).  Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Use baudrate calculated based on sync pattern received from ECU 1 = Use baudrate of ComParam CP_Baudrate	ISO_9141_2 = 0 ISO_14230_2 = 0
CP_5BaudInitBaudrate	Description: Represents the desired bit time (and therefore resulting baudrate) be used for the transmission of the ECU address during a 5-baud initialization sequence.  Type: PDU_PT_UNUM32 Range: [0x0; 0xFFFFFFFF] Resolution: 1 µs	ISO_9141_2 = 200000 ISO_14230_2 = 200000
CP_AccessTiming_Ecu	Description: Timing parameters to be sent/used in response to a Service Id 0x83 (Access Timing Service) with TPI 1, 2 or 3. For a TPI of 1 (set default values), the ECU will set the timing parameters to the default values specified in ISO 14230-2. For a TPI of 2 (read active values), the ECU will return the active timing parameters in the response message. For a TPI of 3 (set parameters), the MVCI protocol module will set the timing parameters to the values to be used by the ECU. This ComParam allows the ECU to define sets of timing parameters to be used for normal and extended timing, as well as override timing values defined by a Tester.  Type: PDU_PT_STRUCTFIELD STRUCTFIELD Format: ComParamStructType= PDU_CPST_ACCESS_TIMING ParamMaxEntries = 8 ParamActEntries = 0 to 8 pStructArray=PDU_PARAM_STRUCT_ACCESS_TIMING	ISO14230_2: ParamActEntries = 0 (not enabled) ISO_14230_4: ParamActEntries = 0 (not enabled)

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_AccessTimingOverride	Description: This ComParam along with CP_ModifyTiming ComParam signals the D-PDU API to override the response from any ECUs to an Access Timing request. The timing parameters are to be used in response to a Service Id 0x83 (Access Timing Service) with TPI 1, 2 or 3. For a TPI of 1 (set default values), the ECU will set the timing parameters to the default values specified in ISO 14230-2. For a TPI of 2 (read active values), the ECU will return the timing parameters in the response message. For a TPI of 3 (set parameters), the MVCI protocol module will set the timing parameters to the values to be used by the ECU. If CP_ModifyTiming is enabled and CP_AccessTimingOverride is not empty (ParamActEntries != 0), then the MVCI protocol module will use data in this ComParam instead of the data returned in a positive response from the ECU for TPI of 2.	ISO_14230_2: ParamActEntries = 0 (not enabled) ISO14230_4: ParamActEntries = 0 (not enabled)
	Type: PDU_PT_STRUCTFIELD STRUCTFIELD Format: ComParamStructType= PDU_CPST_ACCESS_TIMING ParamMaxEntries = 8 ParamActEntries = 0 to 8 pStructArray=PDU_PARAM_STRUCT_ACCESS_TIMING	
CP_Ar	Description: Time for transmission of the CAN frame (any N_PDU) on the receiver side.	ISO_15765_2=1000000 ISO_15765_4=25000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 $\mu$ s	
CP_Ar_Ecu	Description: Time for transmission of the CAN frame (any N_PDU) on the receiver side.	ISO_15765_2=1000000 ISO_15765_4=25000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 $\mu$ s	
CP_As	Description: Time for transmission of the CAN frame (any N_PDU) on the sender side.	ISO_15765_2=1000000 ISO_15765_4=25000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 $\mu$ s	
CP_As_Ecu	Description: Time for transmission of the CAN frame (any N_PDU) on the sender side.	ISO_15765_2=1000000 ISO_15765_4=25000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 $\mu$ s	
CP_BlockSize	Description: This sets the block size that the interface should report to the vehicle for receiving segmented transfers in a Transmit Flow Control Message.	ISO_15765_2=0 ISO_15765_4=0 SAE_J1939_21=0xFF
	Type: PDU_PT_UNUM32 Range: [0; 0xFF] Resolution: 1 Block	SAE_J1708=0xFF

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_BlockSize_Ecu	Description: This sets the block size that the ECU should report to the tester for receiving segmented transfers in a Transmit Flow Control Message.	ISO_15765_2=0 ISO_15765_4=0 SAE_J1939_21=0xFF SAE_J1708=0xFF
	Type: PDU_PT_UNUM32 Range: [0; 0xFF] Resolution: 1 Block	
CP_BlockSizeOverride	Description: This sets the block size that the interface should use to send segmented messages to the vehicle. The flow control value reported by the vehicle should be ignored.	ISO_15765_2=0xFFFF ISO_15765_4=0xFFFF SAE_J1939_21=0xFFFF SAE_J1708=0xFFFF
	Type: PDU_PT_UNUM32 Value: [0; 0xFFFF] 0 - 0xFFFE = Block size 0xFFFF = Use the value reported by the vehicle Resolution: 1 Block NOTE 1 A mismatch between the block size reported by the vehicle and CP_BlockSizeOverride may result in unexpected flow control frames or flow control timeouts (see CP_Bs) where both cases lead to a connection abort.	
CP_Br	Description: Time the tester shall wait before transmitting the next FlowControl. This is equivalent to $T_h/T_{hold}$ in SAE J1939-21. For ISO 15765-2 and ISO 15765-4, this value is an "Override" ComParam allowing for adding a synthetic delay before transmitting a flow control frame which would be transmitted as fast as possible according to ISO 15765-2. For SAE 1939-21: For destination-specific messages, the receiving unit is responsible for coordinating flow control between the nodes. If the receiving unit wants to stop the data flow momentarily while the connection is open, it shall use the CTS message, setting the number of packets equal to zero. When the receiving unit wants to stop the data flow for several seconds, the receiving unit shall send the CTS message once every CP_Br microseconds. For SAE J1708: For destination-specific messages, the receiving unit is responsible for coordinating flow control between the nodes. If the receiving unit wants to stop the data flow momentarily while the connection is open, it shall use the CTS message, setting the number of packets equal to zero. When the receiving unit wants to stop the data flow for several seconds, the receiving unit shall send the CTS message once every CP_Br microseconds.	ISO_15765_2=0 ISO_15765_4=0 SAE_J1939_21=500000 SAE_J1708=500000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 $\mu$ s	
CP_Br_Ecu	Description: Time until transmission of the next FlowControl. This is a performance requirement ComParam.	ISO_15765_2=10000 ISO_15765_4=10000 SAE_J1939_21=500000 SAE_J1708=500000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 $\mu$ s	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_Bs	Description: Timeout until reception of the next FlowControl. This is equivalent to T4 in J1939-21.	ISO_15765_2=1000000 ISO_15765_4=75000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs	SAE_J1939_21=1050000 SAE_J1708=60000000
CP_Bs_Ecu	Description: Timeout until reception of the next FlowControl.	ISO_15765_2=1000000 ISO_15765_4=75000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs	SAE_J1939_21=1050000 SAE_J1708=60000000
CP_CanDataSizeOffset	Description: Offset subtracted from the total number of expected bytes received/transmitted in a first frame message.	ISO_15765_2=0 ISO_15765_4=0
	Type: PDU_PT_UNUM32 Range: [0; 8] Resolution: 1 Byte	
CP_CanFillerByte	Description: Padding data byte to be used to pad all USD1 type transmits frames (SF, FC and last CF).	ISO_15765_2=0x55 ISO_15765_4=0x00
	Type: PDU_PT_UNUM32 Range: [0; 0xFF] NOTE 2 The padding data byte value is typically 0x00, 0x55 or 0xAA.	ISO_J1939_21=0x00 ISO_11898_RAW=0x00
CP_CanFillerByteHandling	Description: Enable Padding, forcing the DLC of a CAN frame to always be 8.	ISO_15765_2=1 ISO_15765_4=1
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Padding Disabled 1 = Padding Enabled	ISO_J1939_21=0 ISO_11898_RAW=0
CP_CanFirstConsecutiveFrameValue	Description: First consecutive frame number to be transmitted/received on a multi-segment transfer. Used to override the normal first consecutive frame value of 1.	ISO_15765_2=1 ISO_15765_4=1
	Type: PDU_PT_UNUM32 Range: [0; 0x0F]	
CP_CanFuncReqExtAddr	Description: Address extension for enhanced diagnostics. The first byte of the requested CAN frame data contains the N_AE/N_TA byte followed by the correct number of PCI bytes. This ComParam is used for all transmitted CAN Frames that have the "Can Address Extension" bit set in the CanIdFormat.	ISO_15765_2=0 ISO_15765_4=0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_CanFuncReqFormat	Description: CAN Format used for a functional address transmit.	ISO_15765_2=0x05 ISO_15765_4=0x05
	Type: PDU_PT_UNUM32 Range: [0; 0x3F] See Table B.15.	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_CanFuncReqId	Description: CAN ID used for a functional address transmit.	ISO_15765_2=0x7DF
	Type: PDU_PT_UNUM32 Range: [0; 0x1FFFFFFF]	ISO_15765_4=0x7DF
CP_CanMaxNumWaitFrames	Description: The maximum number of WAIT flow control frames allowed during a multi-segment transfer. For SAE J1939, this is the maximum number of allowed CTS frames.	ISO_15765_2=255 ISO_15765_4=0 SAE_J1939_21=255
	Type: PDU_PT_UNUM32 Range: [0; 1027]	
CP_CanPhysReqExtAddr	Description: Address extension for enhanced diagnostics. The first byte of the requested CAN frame data contains the N_AE/N_TA byte followed by the correct number of PCI bytes. This ComParam is used for all transmitted CAN Frames that have the "Can Address Extension" bit set in the CanIdFormat.	ISO_15765_2=0x00 ISO_15765_4=0x00
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_CanPhysReqFormat	Description: CAN Format used for a physical address transmit.	ISO_15765_2=0x05
	Type: PDU_PT_UNUM32 Range: [0; 0x3F] See Table B.15.	ISO_15765_4=0x05
CP_CanPhysReqId	Description: CAN ID used for a physical address transmit.	ISO_15765_2=0x7E0
	Type: PDU_PT_UNUM32 Range: [0; 0x1FFFFFFF]	ISO_15765_4 = 0x7E0
CP_CanRespUSDTExtAddr	Description: Extended Address used for a USDT response from an ECU if the CAN Format indicates address extension.	ISO_15765_2=0 ISO_15765_4=0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_CanRespUSDTFormat	Description: CAN Format for the USDT CAN ID received from an ECU (Segment type Bit shall = 1).	ISO_15765_2=0x05 ISO_15765_4=0x05
	Type: PDU_PT_UNUM32 Range: [0; 0xF] See Table B.16.	
CP_CanRespUSDId	Description: Received USDT CAN ID from an ECU.	ISO_15765_2=0x7E8
	Type: PDU_PT_UNUM32 Range: [0; 0x1FFFFFFF, 0xFFFFFFFF] NOTE 3 0xFFFFFFFF indicates that the ComParam is not used. This ComParam is used in the Unique Response Identifier Table for CAN protocols.	ISO_15765_4=0x7E8
CP_CanRespUUDTExtAddr	Description: Extended Address used for UUDT response if the CAN Format indicates address extension.	ISO_15765_2=0 ISO_15765_4=0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	ISO_11898_RAW=0x00

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_CanRespUUDTFormat	Description: Received CAN Format for CAN ID without segmentation (Segment Type Bit shall = 0).	ISO_15765_2=0x00 ISO_15765_4=0x00
	Type: PDU_PT_UNUM32 Range: [0; 0xF] See Table B.17.	ISO_11898_RAW=0x00
CP_CanRespUUDTId	Description: Received UUDT CAN ID from an ECU.	ISO_15765_2=0x FFFFFFFF
	Type: PDU_PT_UNUM32 Range: [0; 0x1FFFFFFFF, 0xFFFFFFFF] NOTE 4 0xFFFFFFFF indicates that the ComParam is not used. This ComParam is used in the Unique Response Identifier Table for CAN protocols.	ISO_15765_4=0x FFFFFFFF ISO_11898_RAW=0x FFFFFFFF
CP_CANFDTxMaxDataLength	Description: Value of ISO15765-2 TX_DL parameter. Also used to identify whether Classic CAN or CANFD to be used for the transmission. TX_DL = 8 : Classic CAN mode or CANFD mode TX_DL > 8 : CANFD mode The value is stored in the D-PDU API, but only used if a proper value for the ComParam CP_CANFDBaudrate is set (can occur after setting CP_CANFDTxMaxDataLength). Otherwise, TX_DL = 8 will be used. NOTE 5 If filler bytes are needed in case of TX_DL > 8, the value defined in CP_CanFillerByte shall be used, regardless of the value in CP_CanFillerByteHandling.	ISO_15765_2 = 0 ISO_11898_RAW = 0
	Type: PDU_PT_UNUM32 Value [0;8;9;10;11;12;13;14;15] 0: TX_DL = 8, Classic CAN 8: TX_DL = 8, CANFD 9: TX_DL = 12, CANFD 10: TX_DL = 16, CANFD 11: TX_DL = 20, CANFD 12: TX_DL = 24, CANFD 13: TX_DL = 32, CANFD 14: TX_DL = 48, CANFD 15: TX_DL = 64, CANFD	
CP_Cr	Description: Timeout for reception of the next ConsecutiveFrame. For SAE J1939-21, this is equivalent to T1.	ISO_15765_2=1000000 ISO_15765_4=150000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs	SAE_J1939_21=750000 SAE_J1708=1000000
CP_Cr_Ecu	Description: Timeout for reception of the next ConsecutiveFrame.	ISO_15765_2=1000000 ISO_15765_4=150000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs	SAE_J1939_21=750000 SAE_J1708=1000000

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_Cs	<p>Description: Time the tester shall wait before transmitting the next Consecutive Frame.</p> <p>For ISO 15765-2 and ISO 15764-4, this value is an "Override" ComParam allowing for adding a synthetic delay before transmitting a consecutive frame which would be transmitted as fast as possible according to ISO 15765-2 in the following cases:</p> <ul style="list-style-type: none"> <li>— if FlowControl is NOT enabled: <ul style="list-style-type: none"> <li>— all consecutive frames,</li> </ul> </li> <li>— if FlowControl is enabled: <ul style="list-style-type: none"> <li>— each consecutive frame following a flow control frame,</li> </ul> </li> </ul> <p>NOTE 6 In this case, STmin is not applicable.</p> <ul style="list-style-type: none"> <li>— consecutive frames following the initial CF of a block of CFs, if the STmin value in the FC=0 and STminOverride=0xFFFFFFFF.</li> </ul> <p>For SAE J1939, this is equivalent to the minimum time between sending packets in a multi-packet broadcast and a multi-packet destination-specific message. See SAE J1939-21:2006, 5.12.3.</p> <p>For SAE J1708, this is equivalent to the minimum time between sending packets in a multi-packet broadcast and a multi-packet destination-specific message.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs</p>	ISO_15765_2=0 ISO_15765_4=0 SAE_J1939_21=50000 SAE_J1708=1000000
CP_Cs_Ecu	<p>Description: Time until transmission of the next Consecutive Frame. (This is used if FC is not enabled or if the STmin value in the FC=0 and STminOverride=0xFFFFFFFF.) See ISO 15765-2. This is a performance requirement ComParam.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs</p>	ISO_15765_2=0 ISO_15765_4=0 SAE_J1939_21=200000 SAE_J1708=1000000
CP_DisableTransportChecksumCheck	<p>Description: Disables the checksum verification when receiving response messages for the protocols where an explicit checksum is transmitted on the transport layer level.</p> <p>Type: PDU_PT_UNUM32 Value[0;1]: 0 = checksum verification is turned ON 1 = checksum verification is turned OFF</p>	ISO_9141_2=0 ISO_14230_2=0 ISO_14230_4=0

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_DoIPLogicalGatewayAddress	Description: The logical address of the DoIP gateway or the DoIP node. For a DoIP gateway, this address has to be used to reach the target addressed in CP_DoIPLogicalEcuAddress or CP_DoIPLogicalFunctionalAddress. For a DoIP node, the values for CP_DoIPLogicalGatewayAddress and CP_DoIPLogicalEcuAddress are identical. NOTE 7 For a DoIP Node, CP_DoIPLogicalGatewayAddress and CP_DoIPLogicalEcuAddress shall be identical.	ISO_13400_2 = 0x0001
	Type: PDU_PT_UNUM32 Range: [0;max] Class: COM	
CP_DoIPLogicalTesterAddress	Description: The logical source address of the Tester.	ISO_13400_2 = 0x0E00
	Type: PDU_PT_UNUM32 Range: [0x0000;0xFFFF] Class: COM	
CP_DoIPLogicalEcuAddress	Description: The logical target address of the ECU to communicate with.	ISO_13400_2 = 0x0001
	Type: PDU_PT_UNUM32 Range: [0x0000;0xFFFF] Class: UNIQUE_ID	
CP_DoIP SecondaryLogicalECUResponse- Address	Description: Secondary logical ECU address delivered with ECU responses corresponding to CAN UUDT addressed responses. The special value of 0 is used to indicate that UUDT responses are not used.	ISO_13400_2 = 0x00
	Type: PDU_PT_UNUM32 Range: [0x0000;0xFFFF] Class: UNIQUE_ID	
CP_DoIPLogicalFunctionalAddress	Description: The logical functional target address to address multiple ECUs behind a DoIP gateway. NOTE 8 Functional addressing is only possible on a single LogicalLink, which leads to the restriction that functional addressing is not possible over multiple DoIP gateways or DoIP nodes.	ISO_13400_2 = 0xE400
	Type: PDU_PT_UNUM32 Range: [0xE400;0xEFFF] (as defined in ISO 13400-2) Class: COM	
CP_DoIPNumberofRetries	Description: The number of retries to be performed when a certain NACK condition is encountered (see Table K.9).	ISO_13400_2 = 0x00
	Type: PDU_PT_UNUM32 Range: [0; 0xFFFFFFFF] Class: COM	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_DoIPDiagnosticAckTimeout	Description: This timeout specifies the maximum time that the test equipment waits for a confirmation ACK or NACK from the DoIP entity after the last byte of a DoIP Diagnostic request message has been sent. After the timeout has elapsed, the request shall be considered lost and the request may be repeated (see CP_RepeatReqCountTrans). (Corresponds to A_DoIP_Diagnostic_Message.)	ISO_13400_2 = 2000000
	Type: PDU_PT_UNUM32 Range: [0; 0xFFFFFFFF] Resolution: 1 µs Class: TIMING	
CP_DoIPRetryPeriod	Description: The period between retries, performed when a certain NACK condition is encountered (see Table K.9).	ISO_13400_2 = 1000000
	Type: PDU_PT_UNUM32 Range: [0; 0xFFFFFFFF] Resolution: 1 µs Class: TIMING	
CP_DoIPRoutingActivationType	Description: Specifies the Activation Type sent in the DoIP Routing Activation request. NOTE 9 TCP connections shall be shared by multiple ComLogicalLinks that share the same value of CP_DoIPRoutingActivationType. For all ComLogicalLinks with differing values of CP_DoIPRoutingActivationType, separate TCP connections (including separate Routing Activations) shall be used.	ISO_13400_2 = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF] Class: COM	
CP_DoIPRoutingActivationTimeout	Description: This ComParam is used to configure the timeout value for a DoIP Routing Activation request. NOTE 10 This timing parameter is based on the time it takes for a DoIP entity to perform an alive check on all its open DoIP sockets. It takes into account the Routing Activation timing, but not the MVCI TCP/IP stack related timings, which is heavily dependent on the underlying operating system and are out of control of the MVCI in regard of their timing behaviour (such as the Windows TCP/IP stack).	ISO_13400_2 = 1000000
	Type: PDU_PT_UNUM32 Range: [0; 125000000] Resolution: 1 µs	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_EcuRespSourceAddress	<p>Description: ECU Source Address response of a non-CAN message. This ComParam is used for response handling only. It is a URID ComParam and is used whether addressing is functional or physical. The protocol handler extracts the ECU address from the response message and uses this information along with other URID ComParams to find a match in the URID table to retrieve the Unique Response Id for the ECU. For physical addressing, it is possible that CP_EcuRespSourceAddress equals CP_PhysReqTargetAddr.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	<p>ISO_9141_2 = 0x10 ISO_14230_2=0x10 ISO_14230_4 = 0x10 SAE_J1850_VPW = 0x10 SAE_J1850_PWM = 0x10</p>
CP_EnableConcatenation	<p>Description: This ComParam instructs the application layer to automatically detect multiple responses from a single ECU and construct a single ECU response to the client application. Only the SID (first byte of the message data) is used to indicate a segmented response to a service request is being sent by the ECU. The application layer will wait for a receive timeout before determining that all responses have been received.</p> <p>e.g. ECU response 1: SID 0x11 0x22 ECU response 2: SID 0x33 0x44 Response to Client application: SID 0x11 0x22 0x33 0x44</p> <p>Type: PDU_PT_UNUM32 Value: [0;1] 0 = Disabled 1 = Enabled</p>	<p>ISO_14230_2 = 0 ISO_14230_4 = 0 ISO_9141_2 = 0 SAE_J1850_VPW = 0 SAE_J1850_PWM = 0</p>
CP_ExtendedTiming	<p>Description: This ComParam is used to define extended timing values for K-Line protocols. The values are used after the key bytes are received from the ECU during the initialization sequence. If CP_ExtendedTiming is not empty (ParamActEntries != 0), then the MVCI protocol module will use data in this ComParam otherwise the MVCI protocol module will use the default extended values defined in ISO 14230-2. For normal timing, the MVCI protocol module uses timing defined in the ComParams: CP_P2Max, CP_P3Min, etc.</p> <p>Type: PDU_PT_STRUCTFIELD STRUCTFIELD Format: ComParamStructType= PDU_CPST_ACCESS_TIMING ParamMaxEntries = 1 ParamActEntries = 0 to 1 pStructArray=PDU_PARAM_STRUCT_ACCESS_TIMING</p>	<p>ISO_14230_2: ParamActLen = 0 (not enabled) ISO_14230_4: ParamActLen = 0 (not enabled)</p>
CP_EscapeSequenceHandling	<p>Description: This ComParam is used to enable/disable the ISO15765-2 multi-frame escape sequence on classical CAN or CAN FD.</p> <p>Type: PDU_PT_UNUM32 Range [0;0xFFFFFFFF] 0: don't use escape sequence 1: use escape sequence according to ISO15765-2 2 - 0xFFFFFFFF: ISO reserved</p>	<p>ISO_15765_2 = 0</p>

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_FillerByte	Description: Padding data byte to be used to pad all SAE J1850, ISO 9141-2 and ISO 14230-4 messages to the full length.	ISO_9141_2 = 0 ISO_14230_4 = 0 ISO_14230_2 = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF] NOTE 11 The padding data byte value is typically 0x00, 0x55 or 0xAA.	SAE_J1850_VPW = 0 SAE_J1850_PWM = 0
CP_FillerByteHandling	Description: Enable Padding for SAE J1850, ISO 9141-2 and ISO 14230-4 messages (see CP_FillerByte).	ISO_9141_2 = 0 ISO_14230_4 = 0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Padding Disabled 1 = Padding Enabled	ISO_14230_2 = 0 SAE_J1850_VPW = 0 SAE_J1850_PWM = 0
CP_FillerByteLength	Description: Length to pad the data portion of the message for SAE J1850, ISO 9141-2 and ISO 14230-4 (see CP_FillerByteHandling and CP_FillerByte).	ISO_9141_2 = 0 ISO_14230_4 = 0 ISO_14230_2 = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF] EXAMPLE If the data payload for the ComPrimitive is 0x21, 0x01 and CP_FillerLength is set to five and CP_FillerByte is set to 0xFF, the data portion of the message would be:  0x21 0x01 0xFF 0xFF 0xFF	SAE_J1850_VPW = 0 SAE_J1850_PWM = 0
CP_FuncReqFormatPriorityType	Description: First Header Byte of a non-CAN message for a functional address transmit. This ComParam is used for proper request message header construction in non-Raw mode.	ISO_9141_2 = 0x68 ISO_14230_2 = 0xC0+n, where n < 64 is generated by the protocol based on the addressing scheme.
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	SAE_J1850_VPW = 0x68 SAE_J1850_PWM = 0x61 ISO_14230_4 = 0xC0+n, where n < 64 is generated by the protocol based on the addressing scheme.
CP_FuncReqTargetAddr	Description: Second Header Byte of a non-CAN message for a functional address transmit. This ComParam is used for proper request message header construction in non-Raw mode.	ISO_9141_2 = 0x6A ISO_14230_2 = 0x33 ISO_14230_4 = 0x33
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	SAE_J1850_VPW = 0x6A SAE_J1850_PWM = 0x6A

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_FuncRespFormatPriorityType	<p>Description: First Header Byte of a non-CAN message received from the ECU for functional addressing. This ComParam is used for response handling only. It is a URID ComParam and is used for functional addressing only. The protocol handler extracts the format/priority byte from the response message and uses this information along with other URID ComParams to find a match in the URID table to retrieve the Unique Response Id for the ECU.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	<p>ISO_9141_2 = 0x48 ISO_14230_2 = 0xC0+n, where n &lt; 64 is generated by the protocol based on the addressing scheme. SAE_J1850_VPW = 0x48 SAE_J1850_PWM = 0x41 ISO_14230_4=0xC0+n, where n &lt; 64 is generated by the protocol based on the addressing scheme.</p>
CP_FuncRespTargetAddr	<p>Description: Second Header Byte of a non-CAN message received from the ECU for functional addressing. This ComParam is used for response handling only. It is a URID ComParam and is used for functional addressing only. The protocol handler extracts the Target address from the response message and uses this information along with other URID ComParams to find a match in the URID table to retrieve the Unique Response Id for the ECU.</p> <p>This information is also used to fill out the functional lookup table for SAE J1850_PWM.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	<p>ISO_9141_2 = 0x6B ISO_14230_2 = 0xF1 ISO_14230_4=0xF1 SAE_J1850_VPW = 0x6B SAE_J1850_PWM = 0x6B</p>
CP_HeaderFormatJ1850	<p>Description: Header Byte configuration to be used for SAE J1850 communication.</p> <p>This setting is used to properly construct the message header bytes to complete the PDU.</p> <p>This ComParam is not used if the ComLogicalLink is in RawMode.</p> <p>Header bytes are constructed following the rules of the protocol specification.</p> <p>Type: PDU_PT_UNUM32 Value: [0; 3] 0 = No Header Bytes 1 = 1 Byte Header 2 = 2 byte Header 3 = 3 byte Header</p>	<p>SAE_J1850_VPW=3-byte-Header SAE_J1850_PWM=2-byte-Header</p>
CP_HeaderFormatKW	<p>Description: Header Byte configuration for K-Line protocols (Keyword).</p> <p>This setting is used to properly construct the message header bytes to complete the PDU.</p> <p>This ComParam is not used if the ComLogicalLink is in RawMode.</p> <p>Header bytes are constructed following the rules of the protocol specification. This ComParam can be used to override any keybyte values received from the ECU during initialization.</p> <p>If the protocol cannot handle the length of a ComPrimitive based on the settings of this ComParam, then an error event, PDU_ERR_EVT_PROT_ERR, is generated and the ComPrimitive is FINISHED.</p>	<p>ISO_9141_2= 0 ISO_14230_2= 0 ISO_14230_4= 0</p>

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
	<p>Type: PDU_PT_UNUM32  Value: [0; 8]  0 = Use the header byte format specified by the ECU key bytes (see NOTE 11)</p> <ul style="list-style-type: none"> <li>1 = 1 Byte Only (max size = 0x3F)</li> <li>2 = 2 Bytes (dependent on length)</li> </ul> <p>if 1st byte &lt;= 0x3F  1st byte = size  2nd byte = not used  else (1st byte &gt; 0x3F)  1st byte does not contain size  2nd byte = size up to 0xFF  End if</p> <p>3 = 2 Bytes always  1st byte never contains size information  2nd byte = size up to 0xFF</p> <p>4 = 3 Bytes Only  1st byte = format with size up to 0x3F  2nd byte = target address  3rd byte = source address</p> <p>5 = 4 Bytes (dependent on length)  if 1st byte &lt;= 0x3F  1st byte = size  2nd byte = target address  3rd byte = source address  4th byte not used.  else (1st byte &gt; 0x3F)  1st byte does not contain size  2nd byte = target address  3rd byte = source address  4th byte = size up to 0xFF  endif</p> <p>6 = 4 Bytes always  1st byte never contains size  2nd byte = target address  3rd byte = source address  4th byte = size up to 0xFF</p> <p>7 = OEM-9141 Header Format (ms nibble of first byte = byte count)  8 = No header bytes</p>	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_InitializationSettings	Description: Set Initialization method.	ISO_9141_2 = 1
	Type: PDU_PT_UNUM32 Value: [1; 3] 1 = 5 Baud Init sequence 2 = Fast Init sequence 3 = No Init sequence	ISO_14230_2 = 2 ISO_14230_4 = 2
CP_ISOKeyByteCount	Description: Used to specify the total number of bytes expected from the ECU in a 5-baud init sequence, following the sync byte (including two key bytes), after which the tester may return the inverted key byte 2.	ISO_9141_2 = 2 ISO_14230_2 = 2
	Type: PDU_PT_UNUM32 Range: [2; 255] Resolution: 1 byte NOTE 12 If e.g. CP_ISOKeyByteCount = 2, the interface accepts 2 key bytes after the sync byte.	
CP_J1939AddrClaimTimeout	Description: Time after sending a Request for Address Claimed before the Tester should send its own Address Claimed message. While waiting for this timeout (and at all times), the tester should be handling Address Claimed messages from all ECUs on the bus.	SAE_J1939_21 = 1250000
	Type: PDU_PT_UNUM32 Range: [0; 20000000] Resolution: 1 µs	
CP_J1939AddressNegotiation-Rule	Description: This ComParam will specify whether the interface should issue a Request for Address Claim upon receiving a STARTCOMM ComPrimitive and under what conditions the interface should send out its own Address Claim.	SAE_J1939 = 0
	Type: PDU_PT_UNUM32 Value: [0; 7] Bit Encoded: Bit 0: 0 = Issue Request for Address Claim upon receiving a STARTCOMM ComPrimitive from client app; 1 = Do not issue Request for Address Claim upon receiving a STARTCOMM ComPrimitive from client app; Bit 1: 0 = Make own Address Claim upon receiving a STARTCOMM ComPrimitive from client app. 1 = Do not make own Address Claim upon receiving a STARTCOMM ComPrimitive from client app. Bit 2: 0 = Respond to a Request for Address Claim message or a challenging Address Claim message from the vehicle bus with own Address Claim. 1 = Do not respond to a Request for Address Claim message or a challenging Address Claim message from the vehicle bus with own Address Claim.	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_J1939DataPage	<p>Description: The data page used to form the data page of a SAE J1939 CAN ID for request messages from the tester to the ECU. This ComParam is used to set bits 24 and 25 of the SAE J1939 CAN ID.</p> <p>Type: PDU_PT_UNUM32 Value: [1; 3] 0 = SAE J1939 Page 0 PGNs 1 = SAE J1939 Page 1 PGNs 2= reserved 3= ISO 15765-3/ISO 14229-3 Addressing Format See Table B.18.</p>	SAE_J1939_21 = 0
CP_J1939MaxPacketTx	<p>Description: Number of frames the tester should request to send at once when sending a RTS. Tester shall be capable of re-sending any block re-requested by the ECU.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 255]</p>	SAE_J1939_21 = 0xFF
CP_J1939Name	<p>Description: Name field from SAE J1939 document. This ComParam will contain the NAME of the Tester. The tester will require this to make an address claim (see SAE J1939-81:2003, 4.1.1 for further details).</p> <p>NOTE 13 If CP_J1939AddressNegotiationRule has Bit 1 and/or Bit 2 set to 0 (protecting an address), a change to this parameter will not take effect until a StartComm CoP is received.</p> <p>Type: PDU_PT_BYTEFIELD BYTEFIELD Format: ParamMaxLen = 8 ParamActLen = 0 to 8 pDataArray = name[8] Range: Each byte = 0x00-0xFF</p>	SAE_J1939_21 = ParamActLen = 0 (not enabled)
CP_J1939Name_Ecu	<p>Description: Name field from SAE J1939 document. This ComParam will contain an ECU NAME.</p> <p>Type: PDU_PT_BYTEFIELD BYTEFIELD Format: ParamMaxLen = 8 ParamActLen = 0 to 8 pDataArray = name[8] Range: Each byte = 0x00-0xFF</p>	SAE_J1939_21 = ParamActLen = 0 (not enabled)
CP_J1939PDUFormat	<p>Description: This ComParam is used to set the PF field (bits 16 to 23) of the SAE J1939 CAN ID for request messages sent from the tester to the ECU.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	SAE_J1939_21 = 0

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_J1939PDUspecific	<p>Description: This ComParam is used to set the PS field (bits 8 to 15) of the SAE J1939 CAN ID. This field is only used if CP_J1939PDUFormat is greater than or equal to 240 (PDU2 format messages). If CP_J1939PDUFormat is less than 240, then the PS field shall be filled with the source address of the ECU that the tester is sending the request to. (Use the source ECU address associated with CP_J1939TargetName.)</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	SAE_J1939_21 = 0
CP_J1939PreferredAddress	<p>Description: List of preferred addresses for the Tester. This ComParam is a list of source addresses for the MVCI protocol module. The first source address claimed by the MVCI Protocol Module remains claimed unless a higher-priority node on the bus requests the same address, at which time the tester will have to try to claim the next address in the list. An address remains claimed until the end of the ComLogicalLink communication. A PDU_COPT_STARTCOMM ComPrimitive will try to claim one of the source addresses in this byte field. Since the MVCI Protocol module might not be able to claim the first address requested, a list of tester addresses are supplied (see SAE J1939-81:2003, 4.1.1 for further details).</p> <p>NOTE 14 If CP_J1939AddressNegotiationRule has Bit 1 and/or Bit 2 set to 0 (protecting an address), a change to this parameter will not take effect until a StartComm CoP is received.</p> <p>Type: PDU_PT_BYTEFIELD BYTEFIELD Format: ParamMaxLen = 8 ParamActLen = 0 to 8 pDataArray = address[8] Range: Each byte = 0x00-0xFF</p>	SAE_J1939_21 = ParamActLen = 0 (not enabled)
CP_J1939PreferredAddress_Ecu	<p>Description: List of preferred addresses for the ECU. This ComParam is a list of source addresses that the ECU would like to acquire on the SAE J1939 bus (see SAE J1939-81:2003, 4.1.1 for further details).</p> <p>Type: PDU_PT_BYTEFIELD BYTEFIELD Format: ParamMaxLen = 2 ParamActLen = 0 to 2 pDataArray=preferredAddrList[2] Range: Each byte = [0; 0xFF]</p>	SAE_J1939_21 = ParamActLen = 0 (not enabled)
CP_J1939SourceAddress	<p>Description: ECU Source Address from a SAE J1939 response message. This ComParam is used for response handling only. It is a URID ComParam. The protocol handler extracts the ECU source address from the response message and uses this information to find a match in the URID table to retrieve the Unique Response Id for the ECU.</p> <p>NOTE 15 This UNIQUE_ID ComParam can also be used to assign a Unique Response Id for a standard, 11-bit Can Id appearing on the vehicle bus.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFFFF]</p>	SAE_J1939 = 0x00

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_J1939SourceName	<p>Description: Name field as described in the SAE J1939 document. This ComParam is used for response handling only. It is the Name of an ECU. The protocol handler will extract the source address from a SAE J1939 response message. By keeping a list of Names and Addresses of all ECUs on the bus (Network Management), the tester will find the Name of the ECU corresponding to the source address extracted from the message and use the Name to find a match in the URID table to retrieve the Unique Response Id for the ECU.</p> <p>Type: PDU_PT_BYTEFIELD            BYTEFIELD Format:            ParamMaxLen = 8            ParamActLen = 0 to 8            pDataArray = name[8]            Range: Each byte = 0x00-0xFE</p>	SAE_J1939_21 = ParamActLen = 0 (not enabled)
CP_J1939TargetAddress	<p>Description: This ComParam is used instead of CP_J1939TargetName in the following three cases:</p> <ul style="list-style-type: none"> <li>a) the ParamActLen for CP_J1939TargetName = 0, or</li> <li>b) the NAME in CP_J1939TargetName is not found in the list of Names and Addresses of the ECUs that have sent out an Address Claim, or</li> <li>c) the Address listed for the NAME is invalid (0xFE).</li> </ul> <p>Type: PDU_PT_UNUM32            Range: [0; 0xFFFF]</p> <p>NOTE 16 If this Parameter = 0xFFFF, the interface will return a failure on the StartComPrimitive. This will serve to inform the client application in case the ECU named in CP_J1939TargetName has not claimed an address or has lost its address to another claimant on the vehicle bus.</p> <p>NOTE 17 If this Parameter = 0xFF and message length &gt; 8, use BAM, else use RTS/CTS.</p>	SAE_J1939_21=0xFFFF
CP_J1939TargetName	<p>Description: Name field from SAE J1939 document. This is the name of the target ECU for a destination-specific outgoing message. Used when CP_J1939PDUFormat &lt; 240; also used in transport protocol when CP_J1939PDUFormat ≥ 240 and message length &gt; 8. By keeping a list of Names and Addresses of all ECUs on the bus (Network Management), the tester will find the ECU Address of the ECU with this name and use it to form the CAN ID. If this ECU has not made an Address Claim on the bus or if the ParamActLen for this ComParam = 0, the tester will use CP_J1939TargetAddress as the destination address (see SAE J1939-81:2003, 4.1.1 for further details).</p> <p>Type: PDU_PT_BYTEFIELD            BYTEFIELD Format:            ParamMaxLen = 8            ParamActLen = 0 to 8            pDataArray = name[8]            Range: Each byte = 0x00-0xFF</p>	SAE_J1939_21 = ParamActLen = 0 (not enabled)
CP_MaxDataLength_Ecu	<p>Description: This ComParam defines the maximum byte length of a segmented USDT message, reported as data length within the first frame (FF_DL).</p> <p>Type: PDU_PT_UNUM32            Range [0; 0xFFFFFFFF]</p>	ISO_15765_2 = 4095

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_MessagePriority	Description: Message Priority SAE J1939 protocol uses the three least significant bits that become part of the CAN ID. This is used only for request messages sent by the tester to the ECU. This parameter is used in bits 26 to 28 of the CAN ID for the SAE J1939 message. SAE J1708: The message priority goes into calculating the required idle bus time before transmitting the message.	SAE_J1939_21 = 6 SAE_J1708 = 8
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_MidReqId	Description: Request Message Identifier used in building a transmit message to an ECU for a SAE J1708 protocol only.	SAE_J1708 = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_MidRespld	Description: Response Message Identifier received from an ECU for a SAE J1708 protocol only.	SAE_J1708 = 0
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_P1Max	Description: Maximum inter-byte time for ECU Responses. Interface shall be capable of handling a P1_MIN time of 0 ms. After the request, the interface shall be capable of handling an immediate response (P2_MIN=0). For subsequent responses, a byte received after P1_MAX shall be considered as the start of the subsequent response.	ISO_9141_2 = 20000 ISO_14230_2 = 20000 ISO_14230_4 = 20000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	
CP_P1Min	Description: This sets the minimum inter-byte time for the ECU responses. Application shall not get or set this value. Interface shall be capable of handling P1_MIN=0. This is a performance requirement ComParam.	ISO_9141_2 = 0 ISO_14230_2 = 0 ISO_14230_4=0
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	
CP_P4Max	Description: Maximum inter-byte time for a tester request.	ISO_9141_2 = 20000 ISO_14230_2 = 20000 ISO_14230_4=20000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	
CP_P4Min	Description: Minimum inter-byte time for tester transmits.	ISO_9141_2 = 5000 ISO_14230_2 = 5000 ISO_14230_4=5000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_PhysReqFormatPriorityType	<p>Description: First Header Byte of a non-CAN message for physical address transmit. This ComParam is used for proper request message header construction in non-Raw mode.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	<p>ISO_9141_2 = 0x6C ISO_14230_2=0x80+n, where n &lt; 64 is generated by the protocol based on the addressing scheme. ISO_14230_4=0x80+n, where n &lt; 64 is generated by the protocol based on the addressing scheme. SAE_J1850_VPW = 0x6C SAE_J1850_PWM = 0xC4</p>
CP_PhysReqTargetAddr	<p>Description: Physical Target Addressing Information used for correct Message Header Construction. This ComParam is used for proper request message header construction in non-Raw mode. It is possible that CP_PhysReqTargetAddr matches CP_EcuRespSourceAddress in one of the URID table entries.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	<p>ISO_9141_2=0x10 ISO_14230_2=0x10 ISO_14230_4=0x10 SAE_J1850_VPW=0x10 SAE_J1850_PWM=0x10 (ECU Variant Specific)</p>
CP_PhysRespFormatPriorityType	<p>Description: First Header Byte of a non-CAN message received from the ECU for physical addressing. This ComParam is used for response handling only. It is a URID ComParam and is used for physical addressing only. The protocol handler extracts the format/priority byte from the response message and uses this information along with other URID ComParams to find a match in the URID table to retrieve the Unique Response Id for the ECU.</p> <p>Type: PDU_PT_UNUM32 Range: [0; 0xFF]</p>	<p>ISO_9141_2=0x6C ISO_14230_2=0x80+n, where n &lt; 64 is generated by the protocol based on the addressing scheme. ISO_14230_4=0x80+n, where n &lt; 64 is generated by the protocol based on the addressing scheme. SAE_J1850_VPW=0x2C SAE_J1850_PWM=0xC4</p>
CP_RepeatReqCountTrans	<p>Description: This ComParam contains a counter to enable a re-transmission of the last request when either a transmit, a receive error or transport layer timeout is detected. This applies to the transport layer only.</p> <p>NOTE 18 For ISO_13400_2 this ComParam is used in case of ACK timeout (see CP_DoIPDiagnosticAckTimeout) and communication errors signalled by IP-Stack / operating system</p> <p>Type: PDU_PT_UNUM32 Range: [0; 255] Resolution: 1 count</p>	All protocols = 0
CP_RequestAddrMode	<p>Description: Addressing Mode to be used for the Com Primitive.</p> <p>Type: PDU_PT_UNUM32 Value: [1; 2] 1 = Use Physical Addressing for the request 2 = Use Functional Addressing for the request</p>	<p>ISO_9141_2 = 2 ISO_15765_2 = 1 ISO_14230_2 = 1 SAE_J1850_VPW=2 SAE_J1850_PWM=2 ISO_15765_4 = 2 ISO_14230_4 = 2</p>

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_SCITransmitMode	Description: SCI transmit mode. Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Transmit using SCI Full duplex mode 1 = Transmit using SCI Half duplex mode	SAE_J2610_SCI = 0
CP_SendRemoteFrame	Description: This ComParam is used for CAN remote frame handling. (No data bytes are transmitted. Just the CAN ID. The first byte of the PDU Data shall contain the Data Length Code.) Type: PDU_PT_UNUM32 Value: [0; 1] 0 = No Remote Frame Transmit 1 = Transmit a Remote Frame using the DLC in the PDU Data	ISO_15765_2 = 0 ISO_15765_4 = 0 SAE_J1939_21 = 0 ISO_11898 = 0
CP_StMin	Description: This sets the separation time the interface should report to the vehicle for receiving segmented transfers in a Transmit Flow Control Message. Type: PDU_PT_UNUM32 Range1: [0x0; 0x7F] Resolution1: 1 ms Range2: [0xF1; 0xF9] Resolution2: 100 µs	ISO_15765_2=0 ISO_15765_4=0
CP_StMin_Ecu	Description: The minimum time the sender shall wait between the transmissions of two ConsecutiveFrame N_PDUs. Type: PDU_PT_UNUM32 Range1: [0x0; 0x7F] Resolution1: 1 ms Range2: [0xF1; 0xF9] Resolution2: 100 µs	ISO_15765_2=0 ISO_15765_4=0
CP_StMinOverride	Description: This sets the separation time the interface should use to transmit segmented messages to the vehicle. The flow control value reported by the vehicle should be ignored. Type: PDU_PT_UNUM32 Value: [0;0xFFFFFFFF] Resolution: 1 µs 0xFFFFFFFF: Use the value reported by the vehicle NOTE 19 This ComParam does only apply if the corresponding CP_CanxxxIdFormat is set to a format value "with FC". For CP_CanxxxIdFormat values "w/o FC", the ComParam CP_Cs applies.	ISO_15765_2=0xFFFFFFFF ISO_15765_4=0xFFFFFFFF
CP_T1Max	Description: This sets the maximum inter-frame response delay. Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	SAE_2610_SCI=20000
CP_T2Max	Description: This sets the maximum inter-frame request delay. Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	SAE_2610_SCI=100000

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_T3Max	Description: This sets the maximum response delay from the ECU after processing a valid request message from the interface. For SAE J1939-21, this is equivalent to Tr.	SAE_J2610_SCI=50000 SAE_J1939_21=200000
	Type: PDU_PT_UNUM32 Range: [0; 2500000] Resolution: 1 µs	
CP_T4Max	Description: This sets the maximum inter-message response delay. For SAE J1939, this is equivalent to T3, the maximum time allowed for the Originator to receive a CTS or an ACK after sending a packet.	SAE_J2610_SCI=20000 SAE_J1939_21=1250000
	Type: PDU_PT_UNUM32 Range: [0; 2500000] Resolution: 1 µs	
CP_T5Max	Description: This sets the maximum inter-message request delay. For SAE J1939, this is equivalent to T2, the maximum time allowed for the Originator to send a packet after receiving a CTS from the Responder.	SAE_J2610_SCI=100000 SAE_J1939_21=1250000
	Type: PDU_PT_UNUM32 Range: [0; 2500000] Resolution: 1 µs	
CP_TesterSourceAddress	Description: Source address of transmitted message for non-CAN messages. This ComParam is used for proper request message header construction in non-Raw mode. This ComParam will also contain the claimed Tester Address for SAE J1939 (see ComParam CP_J1939PreferredAddress), which can be read by the client application after a successful address claim determined during a PDU_COPT_STARTCOMM ComPrimitive.	ISO_9141_2 = 0xF1 ISO_14230_2 = 0xF1 ISO_14230_4 = 0xF1 SAE_J1850_VPW = 0xF1 SAE_J1850_PWM = 0xF1
	Type: PDU_PT_UNUM32 Range: [0; 0xFF]	
CP_TIdle	Description: Minimum bus idle time before tester starts the address byte sequence or the fast init sequence. (TIdle replaces W0 and W5.)	ISO_9141_2 = 300000 ISO_14230_2 = 300000 ISO_14230_4 = 300000
	Type: PDU_PT_UNUM32 Range: [0; 10000000] Resolution: 1 µs	
CP_TInil	Description: Sets the duration for the low pulse in a fast initialization sequence.	ISO_9141_2 = 25000 ISO_14230_2 = 25000 ISO_14230_4 = 25000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_TPCConnectionManagement	Description: When transmitting a message longer than 21 bytes, this tells whether to use a Broadcast message or an RTS/CTS protocol.	SAE_J1708=0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = send the data bytes through broadcast (PID = 192) 1 = send the data bytes using connection mode data transfer (PIDs 197 and 198)	
CP_TWup	Description: Sets total duration of the wakeup pulse (TWUP-TINIL)=high pulse before start communication message.	ISO_9141_2 = 50000 ISO_14230_2 = 50000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 $\mu$ s	ISO_14230_4 = 50000
CP_W1Max	Description: Maximum time from the end of address byte to start of the synchronization pattern from the ECU.	ISO_9141_2 = 300000 ISO_14230_2 = 300000
	Type: PDU_PT_UNUM32 Range: [0; 1000000] Resolution: 1 $\mu$ s	ISO_14230_4 = 300000
CP_W1Min	Description: Minimum time from the end of address byte to start of the synchronization pattern from the ECU.	ISO_9141_2 = 60000 ISO_14230_2 = 60000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 $\mu$ s	ISO_14230_4 = 60000
CP_W2Max	Description: Maximum time from the end of the synchronization pattern to the start of key byte 1.	ISO_9141_2 = 20000 ISO_14230_2 = 20000
	Type: PDU_PT_UNUM32 Range: [0; 1000000] Resolution: 1 $\mu$ s	ISO_14230_4 = 20000
CP_W2Min	Description: Minimum time from the end of the synchronization pattern to the start of key byte 1.	ISO_9141_2 = 5000 ISO_14230_2 = 5000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 $\mu$ s	ISO_14230_4 = 5000
CP_W3Max	Description: Maximum time between key byte 1 and key byte 2.	ISO_9141_2 = 20000 ISO_14230_2 = 20000
	Type: PDU_PT_UNUM32 Range: [0; 1000000] Resolution: 1 $\mu$ s	ISO_14230_4 = 20000
CP_W3Min	Description: Minimum time between key byte 1 and key byte 2.	ISO_9141_2 = 0 ISO_14230_2 = 0
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 $\mu$ s	ISO_14230_4 = 0

Table B.20 — (continued)

Short name	Detailed description	Default (by protocol)
CP_W4Max	Description: Maximum time between receiving key byte 2 from the vehicle and the inversion being returned by the interface. Same is true for the inverted key byte 2 sent by the tester and the received inverted address from the vehicle.	ISO_9141_2 = 50000 ISO_14230_2 = 50000 ISO_14230_4 = 50000
	Type: PDU_PT_UNUM32 Range: [0; 1000000] Resolution: 1 µs	
CP_W4Min	Description: Minimum time between receiving key byte 2 from the vehicle and the inversion being returned by the interface. Same is true for the inverted key byte 2 sent by the tester and the received inverted address from the vehicle.	ISO_9141_2 = 25000 ISO_14230_2 = 25000 ISO_14230_4 = 25000
	Type: PDU_PT_UNUM32 Range: [0; 250000] Resolution: 1 µs	

### B.5.3 ComParam definitions for physical layer

Table B.21 — Physical layer detailed ComParam table

Short name	Description	Structure, resolution	Range (by protocol)	Type	Default (by protocol)
CP_Baudrate	Description: Represents the desired baud rate. If the desired baud rate cannot be achieved within the tolerance of the protocol, the interface will remain at the previous baud rate.				ISO_11898_2_DWCAN=500k ISO_11898_3_DWFTCAN=125k ISO_11992_1_DWCAN=125k ISO_9141_2_UART = 10400 ISO_14230_1_UART = 10400 SAE_J2610_UART = 7812 SAE_J1708_UART = 9600 SAE_J1939_11_DWCAN=250k SAE_J1850_VPW = 10400 SAE_J1850_PWM = 41600 SAE_J2411_SWCAN = 33333
	Type: PDU_PT_UNUM32 Range: [0x0; 0xFFFFFFFF] Resolution: 1 bps				
CP_CANFDBaudrate	Description: Baudrate to be used during the high bit-rate phase of the CANFD frame transmission/reception. A value of "0" indicates that CP_Baudrate value will also be used for the high bitrate of CAN FD. NOTE Value of ComParam CP_CANFDTxMaxDataLength controls whether CANFD is used for TX or not.				ISO_11898_2_DWCAN=0
	Type: PDU_PT_UNUM32 Range: [0x0; 0xFFFFFFFF] Resolution: 1 bps				
CP_BitSamplePoint	Description: This sets the desired bit sample point as a percentage of the bit time.				ISO_11898_2_DWCAN=80 % ISO_11898_3_DWFTCAN=80 % ISO_11992_1_DWCAN=80 % SAE_J1939_11_DWCAN=80 % SAE_J2411_SWCAN=87 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				

Table B.21 — (continued)

Short name	Description	Structure, resolution	Range (by protocol)	Type	Default (by protocol)
CP_BitSamplePoint_Ecu	Description: This sets the desired bit sample point as a percentage of the bit time.				ISO_11898_2_DWCAN = 80 % ISO_11898_3_DWFTCAN=80 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				ISO_11992_1_DWCAN=80 % SAE_J1939_11_DWCAN=80 % SAE_J2411_SWCAN=87 %
CP_CANFDBitSamplePoint	Description: This sets the desired bit sample point as a percentage of the bit time to be used during the high bit-rate phase of the CANFD frame transmission/reception.				ISO_11898_2_DWCAN=80 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				
CP_CANFDBitSamplePoint_Ecu	Description: This sets the desired bit sample point as a percentage of the bit time to be used during the high bit-rate phase of the CANFD frame transmission/reception.				ISO_11898_2_DWCAN=80 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				
CP_CanBaudrateRecord	Description: List of baud rates to use during an OBD CAN initialization sequence.				ISO_11898_2_DWCAN: ParamActLen = 2, pDataArray = {500000, 250000}
	Type: PDU_PT_LONGFIELD LONGFIELD Format: ParamMaxLen = 12 ParamActLen = 0 to 12 pDataArray=BaudrateList[12] Range: Each entry = [0x00000000; 0xFFFFFFFF]				SAE_J1939_11_DWCAN: ParamActLen = 1, pDataArray = {250000}
CP_K_L_LineInit	Description: K and L line usage for ISO 9141 and ISO 14230 initialization address.				ISO_9141_2_UART = 0 ISO_14230_1_UART = 0
	Type: PDU_PT_UNUM32 Range: [0; 2] 0 = Use L-line and K-line for initialization address 1 = Use K-line only for initialization address 2 = Use L-Line only for initialization address				
CP_K_LinePullup	Description: Control the K-Line voltage to either 12 V or 24 V.				ISO_9141_2_UART = 0 ISO_14230_1_UART = 0
	Type: PDU_PT_UNUM32 Value: [0; 2] 0 = No pull-up 1 = 12V 2 = 24V				
CP_ListenOnly	Description: Enable a Listen Only mode on the ComLogicalLink. This will cause the link to no longer acknowledge received frames on the CAN Network.				ISO_11898_2_DWCAN=0 ISO_11898_3_DWFTCAN=0 ISO_11992_1_DWCAN=0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = Listen Only Mode Disabled 1 = Listen Only Mode Enabled				SAE_J1939_11_DWCAN=0 SAE_J2411_SWCAN=0

Table B.21 — (continued)

Short name	Description	Structure, resolution	Range (by protocol)	Type	Default (by protocol)
CP_NetworkLine	Description: This sets the network line(s) that are active during communication (for cases where the physical layer allows this).				SAE_J1850_PWM = 0
	Type: PDU_PT_UNUM32 Value: [0; 2] 0 = BUS_NORMAL 1 = BUS_PLUS 2 = BUS_MINUS				
CP_SamplesPerBit	Description: Number of samples per bit.				ISO_11898_2_DWCAN=0 ISO_11898_3_DWFTCAN=0 ISO_11992_1_DWCAN=0 SAE_J1939_11_DWCAN=0 SAE_J2411_SWCAN=0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = 1sample per bit 1 = 3 samples per bit				
CP_SamplesPerBit_Ecu	Description: Number of samples per bit for the ECU.				ISO_11898_2_DWCAN=0 ISO_11898_3_DWFTCAN=0 ISO_11992_1_DWCAN=0 SAE_J1939_11_DWCAN=0 SAE_J2411_SWCAN = 0
	Type: PDU_PT_UNUM32 Value: [0; 1] 0 = 1sample per bit 1 = 3 samples per bit				
CP_SyncJumpWidth	Description: This sets the desired synchronization jump width as a percentage of the bit time.				ISO_11898_2_DWCAN=15 % ISO_11898_3_DWFTCAN=15 % ISO_11992_1_DWCAN=15 % SAE_J1939_11_DWCAN=15 % SAE_J2411_SWCAN=15 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				
CP_SyncJumpWidth_Ecu	Description: This sets the desired synchronization jump width as a percentage of the bit time.				ISO_11898_2_DWCAN = 15 % ISO_11898_3_DWFTCAN = 15 % ISO_11992_1_DWCAN = 15 % SAE_J1939_11_DWCAN = 15 % SAE_J2411_SWCAN = 15 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				
CP_CANFDSyncJumpWidth	Description: This sets the desired synchronization jump width as a percentage of the bit time to be used during the high bit-rate phase of the CANFD frame transmission/reception.				ISO_11898_2_DWCAN=15 %
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				
CP_CANFDSyncJumpWidth_Ecu	Description: This sets the desired synchronization jump width as a percentage of the bit time to be used during the high bit-rate phase of the CANFD frame transmission/reception.				ISO_11898_2_DWCAN=15%
	Type: PDU_PT_UNUM32 Range: [0; 100] Resolution: 1 %				

Table B.21 — (continued)

Short name	Description	Structure, resolution	Range (by protocol)	Type	Default (by protocol)
CP_TerminationType	Description: CAN termination settings. This ComParam can be used to manually change the termination being used on the vehicle bus line.				ISO_11898_2_DWCAN = 0 ISO_11898_3_DWFTCAN = 0 ISO_11992_1_DWCAN = 0 SAE_J1939_11_DWCAN = 0 SAE_J2411_SWCAN = 0
	Type: PDU_PT_UNUM32 Value: [0; 4] 0 = No termination 1 = AC termination 2 = 60 Ohm termination 3 = 120 Ohm termination 4 = SWCAN termination				
CP_TerminationType_Ecu	Description: CAN termination settings for SWCAN ECU emulation.				SAE_J2411_SWCAN = 0
	Type: PDU_PT_UNUM32 Value: [0, 5; 6] 0 = No termination 5 = SWCAN Unit Load termination (see SAE J2411) 6 = SWCAN Primary Load termination				
CP_UartConfig	Description: Configure the parity, data bit size and stop bits of a Uart protocol.				ISO_9141_2_UART = 06 ISO_14230_1_UART = 06 SAE_J2610_UART = 06 SAE_J1708_UART = 06
	Type: PDU_PT_UNUM32 Value: [0; 17] 00 = 7N1 01 = 7O1 02 = 7E1 03 = 7N2 04 = 7O2 05 = 7E2 06 = 8N1 07 = 8O1 08 = 8E1 09 = 8N2 10 = 8O2 11 = 8E2 12 = 9N1 13 = 9O1 14 = 9E1 15 = 9N2 16 = 9O2 17 = 9E2				

## Annex C (informative)

### D-PDU API manufacturer-specific ComParams and protocols

#### C.1 Manufacturer-specific protocols — Support and naming conventions

##### C.1.1 General

The D-PDU API is not restricted to specific diagnostic protocols. Since the supported protocols and its ComParams are described in the MDF, the protocol support of an MVCI protocol module using the D-PDU API can be extended easily. The only important requirement is that the designation of the protocols (e.g. protocol names) is unique. For standard ComParam and protocol naming guidelines, see Annex B. An attachment containing ComParams is available as an electronic file accompanying this document.

##### C.1.2 Manufacturer protocol naming guidelines

The following naming guidelines apply.

- Customer- or manufacturer-specific protocols shall be named with the prefix MSP\_ (i.e. inshort name for manufacturer-specific protocol) followed by a protocol name, which can be freely defined by the manufacturer (e.g. MSP\_KWP9999\_on\_ISO14230\_1\_UART).
- The protocol short names, which are used in the MDF file for the PROTOCOL element, are a concatenation of the application layer specification name, plus the transport layer specification layer name, connected by the additional string “\_on\_” as shown in Table B.3.
- The physical layer name as shown in Table B.2 is used in the MDF file as short name for the BUSTYPE element.
- When possible, a customer- or manufacturer-specific protocol should try and reuse a layer either from another customer-specified protocol or from one of the standardized protocols.

##### C.1.3 Manufacturer protocol communication parameters (ComParams)

ISO 22901-1 (ODX specification) already defines mechanisms for the description of ComParams for a protocol. The D-PDU API can be used in combination with ODX data files. Therefore, the description mechanisms from the ODX specification are used to define ComParams in the MDF (for details, see the ODX specification).

For each protocol that the MVCI protocol module supports, the MDF shall assign the reference between the ComParams and the unique protocol id. For each protocol, the MDF includes the following elements:

- ProtocolName;
- short name and unique ID for each protocol ComParam.

The format for a manufacturer-specific ComParam shall be CPM\_xxxx\_yyyy, where xxxx is the manufacturer's acronym and yyyy is the parameter name.

**NOTE** For protocol ComParams, only parameter data types supported by the D-PDU API are used. See B.3.3.

For manufacturer-specific protocols, it is recommended to reuse any of the standard protocol ComParams applicable and only add new ComParams that are needed to support the unique features of the manufacturer-specific protocol.

The ID value for each protocol ComParam can be freely assigned by the MVCI supplier because the ID value is used only within the supplier-specific D-PDU API.

The MDF contains the following information about a ComParam:

- short name;
- long name;
- ComParam class;
- layer info;
- ComParam data type;
- minimum value;
- maximum value;
- default value (per protocol).

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

## Annex D (normative)

### D-PDU API constants

#### D.1 Constants

##### D.1.1 D-PDU API item type values

```
typedef enum E_PDU_IT {
    PDU_IT_IO_UNUM32           = 0x1000,    /* IOCTL UNUM32 item. */
    PDU_IT_IO_PROG_VOLTAGE    = 0x1001,    /* IOCTL Program Voltage item. */
    PDU_IT_IO_BYTEARRAY      = 0x1002,    /* IOCTL Byte Array item. */
    PDU_IT_IO_FILTER          = 0x1003,    /* IOCTL Filter item. */
    PDU_IT_IO_EVENT_QUEUE_PROPERTY = 0x1004, /* IOCTL Event Queue Property item. */
    PDU_IT_RSC_STATUS         = 0x1100,    /* Resource Status item */
    PDU_IT_PARAM              = 0x1200,    /* ComParam item */
    PDU_IT_RESULT             = 0x1300,    /* Result item */
    PDU_IT_STATUS             = 0x1301,    /* Status notification item */
    PDU_IT_ERROR              = 0x1302,    /* Error notification item */
    PDU_IT_INFO               = 0x1303,    /* Information notification item */
    PDU_IT_RSC_ID             = 0x1400,    /* Resource ID item */
    PDU_IT_RSC_CONFLICT       = 0x1500,    /* Resource Conflict Item */
    PDU_IT_MODULE_ID          = 0x1600,    /* Module ID item */
    PDU_IT_UNIQUE_RESP_ID_TABLE = 0x1700,   /* Unique Response Id Table Item */
    PDU_IT_IO_VEHICLE_ID_REQUEST = 0x1800,  /* DoIP Vehicle ID Request Item */
    PDU_IT_ETH_SWITCH_STATE   = 0x1801,   /* DoIP Ethernet Activation Item */
    PDU_IT_IO_ENTITY_ADDRESS  = 0x1802,   /* DoIP entity addressing item */
    PDU_IT_IO_ENTITY_STATUS   = 0x1803,   /* DoIP entity status item */
} T_PDU_IT;
```

##### D.1.2 ComPrimitive type values

```
typedef enum E_PDU_COPT {
    PDU_COPT_STARTCOMM = 0x8001,    /* Start communication with ECU by sending an optional request.
    The detailed behaviour is protocol dependent. For certain protocols
    (e.g. ISO 14230), this ComPrimitive is required as the first
    ComPrimitive. This ComPrimitive is also required to put the
    ComLogicalLink into the state PDU_CLLST_COMM_STARTED
    which allows for tester present messages to be enabled (see
    CP_TesterPresentHandling). Once tester present handling is
    enabled the message is sent immediately, prior to the initial tester
    present cyclic time (CP_TesterPresentTime) */
    PDU_COPT_STOPCOMM  = 0x8002,    /* Stop communication with ECU by sending an optional request.
    The detailed behaviour is protocol dependent. After successful
    completion of this ComPrimitive type, the ComLogicalLink is
    placed into PDU_CLLST_ONLINE state and no further tester
    presents will be sent. A PDU_COPT_STARTCOMM ComPrimitive
    might be required by some protocols (e.g. ISO 14230) to begin
    communications again.*/
```

PDU\_COPT\_UPDATEPARAM = 0x8003, /\* Copies ComParams related to a ComLogicalLink from the working buffer to the active buffer. Prior to update, the values need to be passed to the D-PDU API by calling PDUSetComParam, which modifies the ComParams in the working buffer. If the physical ComParams are locked by another ComLogicalLink, then a PDU\_COPT\_UPDATEPARAM will generate an error event (PDU\_ERR\_EVT\_RSC\_LOCKED) if physical ComParams are to be modified.

NOTE 1 If the CLL is in the PDU\_CLLST\_COMM\_STARTED state and tester present handling is enabled (see CP\_TesterPresentHandling) any changes to one of the tester present ComParams will cause the tester present message to be sent immediately, prior to the initial tester present cyclic time.

NOTE 2 Protocol handler always waits the proper P3Min time before allowing any transmit. See CP\_P3Min, CP\_P3Func, CP\_P3Phys.\*/\*

PDU\_COPT\_SENDRECV = 0x8004, /\* Send request data and/or receive corresponding response data (single or multiple responses). See 10.1.4.17 for detailed settings of the PDU\_COP\_CTRL\_DATA structure.\*/\*

PDU\_COPT\_DELAY = 0x8005, /\* Wait the given time span before executing the next ComPrimitive.\*/\*

PDU\_COPT\_RESTORE\_PARAM = 0x8006, /\* Copies ComParams related to a ComLogicalLink from active buffer to working buffer. (Converse functionality of PDU\_COPT\_UPDATEPARAM.)\*/

} T\_PDU\_COPT;

### D.1.3 Object type values

See PDUGetObjectId function

```
typedef enum E_PDU_OBJT {
    PDU_OBJT_PROTOCOL      = 0x8021, /* Object type for object PROTOCOL of MDF.*/*
    PDU_OBJT_BUSTYPE       = 0x8022, /* Object type for object BUSTYPE of MDF.*/*
    PDU_OBJT_IO_CTRL       = 0x8023, /* Object type for object IO_CTRL of MDF.*/*
    PDU_OBJT_COMPARAM      = 0x8024, /* Object type for object COMPARAM of MDF.*/*
    PDU_OBJT_PINTYPE       = 0x8025, /* Object type for object PINTYPE of MDF.*/*
    PDU_OBJT_RESOURCE      = 0x8026, /* Object type for object RESOURCE of MDF.
```

NOTE The caller of this function with this object type would need to know the vendor specific short-name of the resource.\*/\*

} T\_PDU\_OBJT;

### D.1.4 Status code values

Status events are returned in an event item type PDU\_IT\_STATUS.

```
typedef enum E_PDU_STATUS {
/* ComPrimitive status */
    PDU_COPST_IDLE        = 0x8010, /* ComPrimitive is in the CommLogicalLink's ComPrimitive Queue
and has not been acted upon. */
    PDU_COPST_EXECUTING   = 0x8011, /* ComPrimitive has been pulled from the CommLogicalLink's
ComPrimitive Queue and is in an active running state. */
    PDU_COPST_FINISHED    = 0x8012, /* ComPrimitive is finished. * No further event items will be
generated for this ComPrimitive. */
    PDU_COPST_CANCELLED   = 0x8013, /* ComPrimitive was cancelled by a PDUCancelComPrimitive
```

```

        request. No further event items will be generated for this
        ComPrimitive. */
        PDU_COPST_WAITING      = 0x8014,      /* A periodic send ComPrimitive (NumSendCycles > 1) has
        finished its periodic cycle and is waiting for its next cyclic time for
        transmission. */

/* ComLogicalLink status */
        PDU_CLLST_OFFLINE     = 0x8050,      /* ComLogicalLink is in communication state "offline". Refer to
        description of PDUConnect, PDUDisconnect. */
        PDU_CLLST_ONLINE      = 0x8051,      /* ComLogicalLink is in communication state "online". A
        PDU_COPT_STARTCOMM ComPrimitive has not been
        commanded. Refer to description of PDUConnect, PDUDisconnect
        */
        PDU_CLLST_COMM_STARTED = 0x8052,     /* ComLogicalLink is in communication state "communication
        started". A PDU_COPT_STARTCOMM ComPrimitive has been
        commanded. The ComLogicalLink is in a transmit/receive state. */

/* Module status */
        PDU_MODST_READY       = 0x8060,      /* The MVCI protocol module is ready for communication. The
        protocol module has been connected by this D-PDU API
        Session (see PDUModuleConnect).*/
        PDU_MODST_NOT_READY   = 0x8061,      /* The MVCI protocol module is not ready for communication.
        Additional information about the cause may be provided via an
        additional vendor specific status code returned in pExtraInfo. Refer
        to description of PDUGetStatus.
        EXAMPLE After running a PDU_IOCTL_RESET command on
        the module, it may take some time for the module until it becomes
        ready. Module is connected by this D-PDU API Session, but it is
        not ready for communication. */
        PDU_MODST_NOT_AVAIL   = 0x8062,      /* The MVCI protocol module is unavailable for connection.
        EXAMPLE Communication was lost after previously being in a
        PDU_MODST_READY state.*/
        PDU_MODST_AVAIL       = 0x8063,      /* The MVCI protocol module is available for connection (i.e. not
        yet connected by a D-PDU API session). (See
        PDUModuleConnect and PDUModuleDisconnect.) */
} T_PDU_STATUS;

```

### D.1.5 Information event values

Information events are returned in an event item type PDU\_IT\_INFO.

```

typedef enum E_PDU_INFO {
        PDU_INFO_MODULE_LIST_CHG      = 0x8070,      /* New MVCI protocol module list is available. Client
        application should call PDUGetModuleIds to get a list of the
        new set of modules and status. This event item is not
        generated when the status of a module changes. Related to
        the System Callback. */
        PDU_INFO_RSC_LOCK_CHG        = 0x8071,      /* There has been a change in the lock status on a shared
        physical resource. Call PDUGetResourceStatus to get a
        description of the new lock status. Only applicable to a
        resource shared by multiple ComLogicalLinks. Related to the
        ComLogicalLink Callback. */
        PDU_INFO_PHYS_COMPARAM_CHG   = 0x8072      /* There has been a change to the physical ComParams by
        another ComLogicalLink sharing the resource. Related to the

```

ComLogicalLink Callback. \*/

} T\_PDU\_INFO;

### D.1.6 Resource status values

Used for element “PDUResourceStatus” of structure PDU\_RSC\_STATUS\_DATA (see 10.1.4.4). See 10.1.3 for interface definition.

**Table D.1 — Resource status values (bit encoded)**

Bit position	Name	Description
0	Usage Status	0 = Resource not in use (default) 1 = Resource in use
1	Availability Status	0 = Resource available (default) 1 = Resource not available
2	Transmit Queue Lock Status	0 = Transmit Queue is not locked (default) 1 = Transmit Queue is locked by a CLL. No other CLL except the one which holds the lock is allowed to transmit on the physical resource.
3	Physical ComParam Lock Status	0 = Physical ComParams are not locked (default) 1 = Physical ComParams are locked by a CLL. No other CLL except the one which holds the lock is allowed to change the physical ComParams for the resource.

### D.1.7 Resource lock values

Used for API functions “PDUUnlockResource” and “PDUUnlockResource”. See 10.1.3 for interface definition.

**Table D.2 — Resource lock/unlock values (bit encoded)**

Bit position	Name	Description
0	Lock Physical ComParams	A ComLogicalLink requests exclusive privilege to modify physical ComParams for a physical resource. No other ComLogicalLink that is sharing the physical resource may attempt to modify the physical ComParams.
1	Lock Physical Transmit Queue	A ComLogicalLink requests exclusive privilege to transmit on a physical resource. No other ComLogicalLink that is sharing the physical resource may transmit any ComPrimitives on the physical resource. Only monitoring of the vehicle bus may be done by other ComLogicalLinks (receive only ComPrimitives).

### D.1.8 Event callback data values

See EventCallback prototype.

```
typedef enum E_PDU_EVT_DATA {
    PDU_EVT_DATA_AVAILABLE = 0x0801, /* This event indicates that there is event data available to be read
    by the application. The data could be an error, status, or result
    item. The application shall call PDUGetEventItem to retrieve the
    item. */
    PDU_EVT_DATA_LOST = 0x0802 /* This event indicates that the Com Logical Link has lost data due
    to a buffer (queue) overrun. No event data is stored in the event
    queue. This is for information only. */
} T_PDU_EVT_DATA.
```

## D.1.9 Reserved ID and handle values

Table D.3 — Reserved ID and handle values

Constant name	Constant value	Description
PDU_ID_UNDEF	0xFFFFFFFFE	Undefined ID value. Used to indicate an ID value is undefined.
PDU_HANDLE_UNDEF	0xFFFFFFFFF	Undefined handle value. Used to indicate a Handle value is undefined.

## D.1.10 IOCTL filter types values

```
typedef enum E_PDU_FILTER {
    PDU_FLT_PASS          = 0x00000001,    /* Allows matching messages into the receive event queue. For all
                                           protocols. */
    PDU_FLT_BLOCK        = 0x00000002,    /* Keeps matching messages out of the event queue. For all
                                           protocols. */
    PDU_FLT_PASS_UUDT    = 0x00000011,    /* Allows matching messages into the receive event queue which
                                           are of a UUDT type only. For ISO 15765 only. */
    PDU_FLT_BLOCK_UUDT   = 0x00000012    /* Keeps matching messages out of the event queue which are of a
                                           UUDT type only. For ISO 15765 only. */
} T_PDU_FILTER;
```

## D.1.11 IOCTL event queue mode type values

```
typedef enum E_PDU_QUEUE_MODE {
    PDU_QUE_UNLIMITED    = 0x00000000,    /* An attempt is made to allocate memory for every item being
                                           placed on the event queue. In Unlimited Mode, the QueueSize is
                                           ignored (Default Mode for a ComLogicalLink). */
    PDU_QUE_LIMITED      = 0x00000001,    /* When the ComLogicalLink's event queue is full (i.e. maximum
                                           size has been reached), no new items are placed on the event
                                           queue. The event items are discarded in this case. */
    PDU_QUE_CIRCULAR     = 0x00000002    /* When the ComLogicalLink's event queue is full (i.e. maximum
                                           size has been reached), then the oldest event item in the queue is
                                           deleted so that the new event item can then be placed in the event
                                           queue. */
} T_PDU_QUEUE_MODE;
```

## D.2 Flag definitions

### D.2.1 TxFlag definition

The TxFlag information is used in the PDU\_COP\_CTRL\_DATA structure (see 10.1.4.17) as part of the function PDUStartComPrimitive function.

Default number of bytes is 4.

**Table D.4 — TxFlag**

BytePos	Bit Pos	Definition	Description	Value
0	7	Unused		
0	6	SUPPRESS_POS_RESP	ISO 15765-3/ISO14229-3 Suppress Positive Response	0 = Not Enabled 1 = Enabled
0	5	ENABLE_EXTRA_INFO	Enable adding header and footer information into the result data (see 10.1.4.11.4). Extra information can be used for ECU response debugging.	0 = Not Enabled 1 = Enabled
0	4-0	Unused		
1	7-0	Reserved		
2	7-3	Reserved		
2	2	Unused		
2	1	WAIT_P3_MIN_ONLY	RAW_MODE Only Modified message timing for ISO 14230. Used to decrease programming time if application knows only one response will be received. Does not affect timing on responses to functional requests.	0 = Interface message timing as specified in ISO 14230 1 = After a response is received for a physical request, the wait time shall be reduced to P3_MIN.
2	0	CAN_29BIT_ID	RAW_MODE Only CAN ID type for ISO 11898, SAE J1939 and ISO 15765. CAN ID is contained in the first 4 bytes of the PDU Data.	0 = 11-bit 1 = 29-bit
3	7	ISO15765_ADDR_TYPE	RAW_MODE Only ISO 15765-2 Addressing Method CAN Extended Address is contained in the byte following the CAN ID in the PDU Data.	0 = no extended address 1 = extended addressing is used
3	6	ISO15765_FRAME_PAD	RAW_MODE Only ISO 15765-2 Frame Padding	0 = no padding 1 = pad all messages to a full CAN frame using the value in the ComParam CP_CanFillerByte
3	5-0	Reserved		

## D.2.2 RxFlag definition

The RxFlag information is used in the PDU\_RESULT\_DATA structure (see 10.1.4.11.4), which is used in a PDU\_IT\_RESULT event item.

Default number of bytes is 4.

**Table D.5 — RxFlag**

Byte Pos	Bit Pos	Definition	Description	Value
0	7	REMOTE_FRAME	CAN remote frame detected. No data bytes are received. The first byte of the D-DPU will contain the data length code.	0 = No Remote Frame Received 1 = Received a Remote Frame
0	6-0	Unused		
1	7-3	Reserved		
1	2	SPD_CHG_EVENT	Indicates that the serial bus has transitioned to a new speed. All communication after this event will occur at the new speed. The message data in this message may contain the monitored Change Speed message received on the serial bus.	0 = No Event 1 = Transitioned to new speed rate
1	1	ECU_TIMING_CHANGE	The timing ComParams values have been modified for the ComLogicalLink. The MSCI protocol module has received a positive timing change message by an ECU in response to a timing change request message (protocol specific). This flag will only be set if the CP_ModifyTiming ComParam is set to Enable.	0 = No Timing Change 1 = Timing ComParams have been modified
1	0	SW_CAN_HV_RX <sup>a</sup>	Indicates that the Single Wire CAN message received was a high-voltage message.	0 = Normal Message 1 = High-Voltage Message
2	7-1	Reserved		
2	0	CAN_29BIT_ID	RAW_MODE ONLY CAN ID type for ISO 11898, SAE J1939 and ISO 15765 CAN ID is contained in the first 4 bytes of the PDU Data.	0 = 11-bit 1 = 29-bit
3	7	ISO15765_ADDR_TYPE	RAW_MODE ONLY ISO 15765-2 Addressing Method CAN Extended Address byte is contained in the bytes following the CAN ID in the PDU Data.	0 = no extended address 1 = extended addressing is used

<sup>a</sup> A SW-CAN transceiver does not provide the capability to determine a high-voltage reception during normal mode of operation and to tie this information to an ongoing reception. A SW-CAN transceiver only provides the capability to indicate a wake-up on the RxD pin during sleep mode operation without being able to receive the CAN frame that forced the wake-up because only the voltage level over a period of time is used to determine a high-voltage signal during this mode of operation. In order to determine a high-voltage reception during normal mode of operation (when CAN messages are received), the voltage level of the bus pin shall be measured in parallel to the CAN message reception and this information shall be tied together according to SAE J2411. Since the information of receiving a high-voltage message is of secondary kind when the MSCI works as a tester device and a separate hardware is required to determine the voltage level, the handling of this flag is optional. In case an MSCI does not have the hardware to measure the SW-CAN bus pin voltage level, this flag shall always be set to "0" = normal message.

Table D.5 — (continued)

Byte Pos	Bit Pos	Definition	Description	Value
3	6	CAN_SEGMENTATION	RAW_MODE ONLY ISO 15765-2 Can Segmentation handling Received message was either handled as a segmented or unsegmented message. (If Segmented, then the segment information was removed from the PDU Data.)	0 = no segmentation 1 = segmented
3	5	Reserved		
3	4	ISO15765_PADDING_ERROR	RAW_MODE ONLY For Protocol ISO 15765, a CAN frame was received with less than 8 data bytes.	0 = No Error 1 = Padding Error
3	3	TX_INDICATION	TxDone indication	0= No TxDone 1= TxDone
3	2	RX_BREAK	SAE J2610 and SAE J1850 VPW only Break indication received	0 = No break received 1 = Break received
3	1	START_OF_MESSAGE	Indicates the reception of the first byte of an ISO 9141 or ISO 14230 message or first frame of an ISO 15765 multiframe message.	0 = Not a start of message indication 1 = First byte or frame received
3	0	TX_MSG_TYPE	Receive Indication/Transmit Loopback	0 = received (i.e. this message was transmitted on the bus by another node) 1 = transmitted (i.e. this is the echo of the message transmitted by the device)

<sup>a</sup> A SW-CAN transceiver does not provide the capability to determine a high-voltage reception during normal mode of operation and to tie this information to an ongoing reception. A SW-CAN transceiver only provides the capability to indicate a wake-up on the RxD pin during sleep mode operation without being able to receive the CAN frame that forced the wake-up because only the voltage level over a period of time is used to determine a high-voltage signal during this mode of operation. In order to determine a high-voltage reception during normal mode of operation (when CAN messages are received), the voltage level of the bus pin shall be measured in parallel to the CAN message reception and this information shall be tied together according to SAE J2411. Since the information of receiving a high-voltage message is of secondary kind when the MVCI works as a tester device and a separate hardware is required to determine the voltage level, the handling of this flag is optional. In case an MVCI does not have the hardware to measure the SW-CAN bus pin voltage level, this flag shall always be set to "0" = normal message.

**D.2.3 CllCreateFlag definition**

The CllCreateFlag information is used in the PDUCreateComLogicalLink function.

Default number of bytes is 4.

Table D.6 — CllCreateFlag

Byte Pos	Bit Pos	Definition	Description	Value
0	7	RawMode	Enables the ability to pass through entire received messages, unchanged, through the datalink (transmitted and received). This feature is protocol specific.	
			RawMode=OFF: When transmitting a message, RawMode OFF indicates that the D-PDU API will add the header bytes and checksums to the pCopData of the ComPrimitive before transmission. When receiving a message, RawMode OFF indicates that the D-PDU API will strip the header bytes and checksums before returning the Result Item (the TxFlag ENABLE_EXTRA_INFO can be used to obtain additional message header/footer information).	0 = OFF (If a protocol is configured to generate header byte information, header bytes will be appended before transmission and stripped after receipt.)
			RawMode=ON: When transmitting a message, RawMode ON indicates that the header bytes and checksums were in the pCopData when PDUStartComPrimitive( ) was called. When receiving a message, RawMode ON indicates that the header bytes and checksums will be left in the Result Item that is returned. The default value of the flag is RawMode=OFF (equal '0'), which means headers and checksums will not be appended to the message.	1 = ON (Interface will pass through the received PDU data. No header bytes will be generated or checked. Only the network layer, such as ISO 15765, will still be enabled.)
0	6	ChecksumMode	For protocols that use checksums, the D-PDU API can create and append the checksum to transmit messages based on this flag.	
			ChecksumMode=OFF This flag is ignored for protocols that do not use checksums. This flag is ignored if RawMode is set to OFF. The default value of the flag is ChecksumMode=OFF (equal '0'), which means a checksum will not be appended to the message.	0 = OFF The D-PDU API will not append a checksum to the transmitted PDU data nor will it validate and remove the received checksum.
			ChecksumMode=ON	1 = ON For the UART protocols using checksums, the D-PDU API will append a checksum to the transmitted PDU data and it will validate and remove the received checksum.
0	5-0	Unused		
1	7-0	Unused		
2	7-0	Unused		
3	7-0	Unused		

#### D.2.4 TimestampFlag definition

The TimestampFlag information is used in the PDU\_RESULT\_DATA structure (see 10.1.4.11.4) as part of the data information returned in a PDU\_IT\_RESULT item.

Default number of bytes is 4.

Table D.7 — TimestampFlag

Byte Pos	Bit Pos	Definition	Description	Value
0	7	TxMsgDoneTimestamp-Indicator	Transmit Done Timestamp Indicator. Indication that the Transmit Done Timestamp value in the PDU_RESULT_DATA structure is valid.	0 = Not Valid 1 = Valid
0	6	StartMsgTimestamp-Indicator	Start Message Timestamp Indicator. Indication that the Start Message Timestamp value in the PDU_RESULT_DATA structure is valid.	0 = Not Valid 1 = Valid
0	5-0	Unused		
1	7-0	Unused		
2	7-0	Unused		
3	7-0	Unused		

### D.3 Function return values

The standard return values cover all return values described in 8.4. For each value, the symbolic definition is provided.

```

typedef enum E_PDU_ERROR {
    PDU_STATUS_NOERROR                = 0x00000000,    /* No error for the function call */
    PDU_ERR_FCT_FAILED                 = 0x00000001,    /* Function call failed (generic failure) */
    PDU_ERR_RESERVED_1                = 0x00000010,    /* Reserved by ISO 22900-2 */
    PDU_ERR_COMM_PC_TO_VCL_FAILED     = 0x00000011,    /* Communication between host and MVCI protocol
    module failed */
    PDU_ERR_PDUAPI_NOT_CONSTRUCTED    = 0x00000020,    /* The D-PDU API has not yet been constructed */
    PDU_ERR_SHARING_VIOLATION         = 0x00000021,    /* A PDUdeconstruct was not called before another
    PDUConstruct */
    PDU_ERR_RESOURCE_BUSY              = 0x00000030,    /* the requested resource is already in use.*/
    PDU_ERR_RESOURCE_TABLE_CHANGED    = 0x00000031,    /* Not used by the D-PDU API */
    PDU_ERR_RESOURCE_ERROR             = 0x00000032,    /* Not used by the D-PDU API */
    PDU_ERR_CLL_NOT_CONNECTED         = 0x00000040,    /* The ComLogicalLink cannot be in the
    PDU_CLLST_OFFLINE state to perform the
    requested operation. */
    PDU_ERR_CLL_NOT_STARTED           = 0x00000041,    /* The ComLogicalLink must be in the
    PDU_CLLST_COMM_STARTED state to perform
    the requested operation. */
    PDU_ERR_INVALID_PARAMETERS        = 0x00000050,    /* One or more of the parameters supplied in the
    function are invalid. */
    PDU_ERR_INVALID_HANDLE            = 0x00000060,    /* One or more of the handles supplied in the
    function are invalid. */
    PDU_ERR_VALUE_NOT_SUPPORTED       = 0x00000061,    /* One of the option values in PDUConstruct is
    invalid. */
    PDU_ERR_ID_NOT_SUPPORTED           = 0x00000062,    /* IOCTL command id not supported by the
    implementation of the D-PDU API */
    PDU_ERR_COMPARAM_NOT_SUPPORTED    = 0x00000063,/* ComParam id not supported by the
    implementation of the D-PDU API */
    PDU_ERR_COMPARAM_LOCKED           = 0x00000064,    /* Physical ComParam cannot be changed because
    it is locked by another ComLogicalLink. */
    PDU_ERR_TX_QUEUE_FULL             = 0x00000070,    /* The ComLogicalLink's transmit queue is full; the
    ComPrimitive could not be queued. */

```

PDU_ERR_EVENT_QUEUE_EMPTY	= 0x00000071,	/* No more event items are available to be read from the requested queue. */
PDU_ERR_VOLTAGE_NOT_SUPPORTED	= 0x00000080,	/* The voltage value supplied in the IOCTL call is supported by the MVCI protocol module. */
PDU_ERR_MUX_RSC_NOT_SUPPORTED	= 0x00000081,	/* The specified pin / resource are not supported by the MVCI protocol module for the IOCTL call. */
PDU_ERR_CABLE_UNKNOWN	= 0x00000082,	/* The cable attached to the MVCI protocol module is of an unknown type. */
PDU_ERR_NO_CABLE_DETECTED	= 0x00000083,	/* No cable is detected by the MVCI protocol module */
PDU_ERR_CLL_CONNECTED	= 0x00000084,	/*The ComLogicalLink is already in the PDU_CLLST_ONLINE state. */
PDU_ERR_TEMPPARAM_NOT_ALLOWED	= 0x00000090,	/* Physical ComParams cannot be changed as a temporary ComParam. */
PDU_ERR_RSC_LOCKED	= 0x000000A0,	/* The resource is already locked.*/
PDU_ERR_RSC_LOCKED_BY_OTHER_CLL	= 0x000000A1,	/* The ComLogicalLink's resource is currently locked by another ComLogicalLink. */
PDU_ERR_RSC_NOT_LOCKED	= 0x000000A2,	/* The resource is already in the unlocked state. */
PDU_ERR_MODULE_NOT_CONNECTED	= 0x000000A3,	/*The module is not in the PDU_MODST_READY state. */
PDU_ERR_API_SW_OUT_OF_DATE	= 0x000000A4,	/*The API software is older than the MVCI protocol module Software*/
PDU_ERR_MODULE_FW_OUT_OF_DATE	= 0x000000A5,	/* The MVCI protocol module software is older than the API software. */
PDU_ERR_PIN_NOT_CONNECTED	= 0x000000A6	/* The requested Pin is not routed by supported cable */
PDU_ERR_IP_PROTOCOL_NOT_SUPPORTED	= 0x000000B0,	/* IP protocol is not supported: e.g. IPv6 used as protocolVersion, but OS doesn't support IPv6 (or it is disabled) */
PDU_ERR_DOIP_ROUTING_ACTIVATION_FAILED	= 0x000000B1,	/* DoIP Routing activation failed */
PDU_ERR_DOIP_ROUTING_ACTIVATION_AUTHENTICATION_FAILED	= 0x000000B2,	/* DoIP Routing activation denied due to missing authentication */
PDU_ERR_DOIP_AMBIGUOUS_LOGICAL_ADDRESS	= 0x000000B3,	/* denied to connect a DoIP LogicalLink with a logical address which is identical for multiple DoIP entities inside a DoIP MVCI module representing a collection of DoIP entities */
PDU_ERR_DOIP_ROUTING_ACTIVATION_INVALID_SOURCE_ADDRESS	= 0x000000B4,	/* Routing activation denied due to response code 0x00 indicating an unknown or invalid source address */
PDU_ERR_DOIP_ROUTING_ACTIVATION_NO_DATA_SOCKET_AVAILABLE	= 0x000000B5,	/* Routing activation denied due to response code 0x01 indicating too many registered and active TCP clients */
PDU_ERR_DOIP_ROUTING_ACTIVATION_SOURCE_ADDRESS_CHANGED	= 0x000000B6,	/* Routing activation denied due to response code 0x02 indicating that a different source address has already been activated before on this TCP connection */
PDU_ERR_DOIP_ROUTING_ACTIVATION_SOURCE_ADDRESS_IN_USE	= 0x000000B7,	/* Routing activation denied due to response code 0x03 indicating that the same source address is already registered and active on a different TCP connection */
PDU_ERR_DOIP_ROUTING_ACTIVATION_CONFIRMATION_REJECTED	= 0x000000B8,	/* Routing activation denied due to response code 0x05 indicating a rejected confirmation */

```

PDU_ERR_DOIP_ROUTING_ACTIVATION_TYPE_UNSUPPORTED = 0x000000B9, /* Routing activation denied
due to response code 0x06 indicating an
unsupported routing activation type */
PDU_ERR_DOIP_ROUTING_ACTIVATION_RESPONSE_CODE_UNKNOWN = 0x000000BA, /* Routing activation
denied due to an unknown (i.e. manufacturer-
specific) routing activation response code */
PDU_ERR_DOIP_ROUTING_ACTIVATION_RESPONSE_TIMEOUT = 0x000000BB /* Time out while waiting for
routing activation response */
PDU_ERR_DOIP_RESPONSE_TIMEOUT = 0x000000BC, /* DoIPCtrlTimeout has occurred */
} T_PDU_ERROR;

```

## D.4 Event error codes

### D.4.1 Error event code returned in PDU\_IT\_ERROR

The standard error codes cover asynchronous error situations, which occur with typical MVCI protocol modules and protocols. Error event codes are returned in an event item type PDU\_IT\_ERROR.

```

typedef enum E_PDU_ERR_EVT {
    PDU_ERR_EVT_NOERROR          = 0x00000000, /* No Error. Event type only returned on a
PDUGetLastError if there were no previous errors for the
requested handle */
    PDU_ERR_EVT_FRAME_STRUCT     = 0x00000100, /* CLL/CoP Error: The structure of the received protocol
frame is incorrect (e.g. wrong frame number, missing FC
...). */
    PDU_ERR_EVT_TX_ERROR        = 0x00000101, /* CLL/CoP Error: Error encountered during transmit
of a ComPrimitive PDU. */
    PDU_ERR_EVT_TESTER_PRESENT_ERROR = 0x00000102, /* CLL/CoP Error: Error encountered in transmitting
a Tester Present message or in receiving an
expected response to a Tester Present message. */
    PDU_ERR_EVT_RSC_LOCKED     = 0x00000109, /* CLL Error: A physical ComParam was not set
because of a physical ComParam lock. */
    PDU_ERR_EVT_RX_TIMEOUT     = 0x00000103, /* CLL/CoP Error: Receive timer (e.g. P2Max)
expired with no expected responses received from
the vehicle. */
    PDU_ERR_EVT_RX_ERROR       = 0x00000104, /* CLL/CoP Error: Error encountered in receiving a
message from the vehicle bus (e.g. checksum error
...). */
    PDU_ERR_EVT_PROTOCOL_ERROR = 0x00000105, /* CLL/CoP Error: Protocol error encountered during
handling of a ComPrimitive (e.g. if the protocol
cannot handle the length of a ComPrimitive). */
    PDU_ERR_EVT_LOST_COMM_TO_VCI = 0x00000106, /* Module Error: Communication to a MVCI protocol
module has been lost. */
    PDU_ERR_EVT_VCI_HARDWARE_FAULT = 0x00000107, /* Module Error: The MVCI protocol module has
detected a hardware error. */
    PDU_ERR_EVT_INIT_ERROR     = 0x00000108, /* CLL/CoP Error: A failure occurred during a
protocol initialization sequence. */
} T_PDU_ERR_EVT;

```

### D.4.2 Additional error code returned in PDU\_IT\_ERROR

The D-PDU API allows for vendor defined additional error codes to be returned in an error event item. All additional error codes and their text translations shall be provided in the MDF.

Table D.8 — Event error and examples for additional error codes

Standard event error codes	Examples for additional error codes for a MDF
<b>PDU_ERR_EVT_FRAME_STRUCT</b>	
PDU_ERR_EVT_FRAME_STRUCT	PDU_XTRA_ERR_ISO15765_PADDING
	PDU_XTRA_ERR_ISO15765_FRAME_NUM
	PDU_XTRA_ERR_1850_FRAME_STRUCT
<b>PDU_ERR_EVT_TX_ERROR</b>	
PDU_ERR_EVT_TX_ERROR	PDU_XTRA_ERR_TX_LOST_ARB
	PDU_XTRA_ERR_TX_NETWORK_FAULT
	PDU_XTRA_ERR_1850_CRC
	PDU_XTRA_ERR_1850_TX_ERR
	PDU_XTRA_ERR_1850_TX_LOST_ARB
	PDU_XTRA_ERR_MAX_WAIT_FRAME
<b>PDU_ERR_EVT_TESTER_PRESENT_ERROR</b>	
PDU_ERR_EVT_TESTER_PRESENT_ERROR	PDU_XTRA_ERR_TP_TX
	PDU_XTRA_ERR_TP_RX
<b>PDU_ERR_EVT_RX_TIMEOUT</b>	
PDU_ERR_EVT_RX_TIMEOUT	PDU_XTRA_ERR_ISO15765_CF_TIMEOUT
	PDU_XTRA_ERR_ISO15765_FC_TIMEOUT
	PDU_XTRA_ERR_RX_TIMEOUT_NO_RSP
<b>PDU_ERR_EVT_RX_ERROR</b>	
PDU_ERR_EVT_RX_ERROR	PDU_XTRA_ERR_CAN_BUS_OFF
	PDU_XTRA_ERR_CAN_BUS_ERROR_ACTIVE
	PDU_XTRA_ERR_CAN_BUS_ERROR_PASSIVE
	PDU_XTRA_ERR_RX_ERRORFRAME
	PDU_XTRA_ERR_CAN_NO_ACK
	PDU_XTRA_ERR_CAN_CRC
	PDU_XTRA_ERR_CAN_BIT_STUFF
	PDU_XTRA_ERR_CAN_BUS_FAULT
	PDU_XTRA_ERR_RX_CHECKSUM_BAD
	PDU_XTRA_ERR_RX_FRAMING
	PDU_XTRA_ERR_RX_NETWORK_FAULT
	PDU_XTRA_ERR_1850_CRC
	PDU_XTRA_ERR_1850_NO_IFR
	PDU_XTRA_ERR_1850_BIT_TIMING
<b>PDU_ERR_EVT_PROT_ERR</b>	
PDU_ERR_EVT_PROT_ERR	PDU_XTRA_ERR_LOST_MASTER
	PDU_XTRA_ERR_NO_POLLING_MSG
	PDU_XTRA_ERR_UNEXP_RESPONSE

Table D.8 — (continued)

Standard event error codes	Examples for additional error codes for a MDF
<b>PDU_ERR_EVT_LOST_COMM_TO_VCI</b>	
PDU_ERR_EVT_LOST_COMM_TO_VCI	PDU_XTRA_ERR_NO_RESP
	PDU_XTRA_ERR_WLAN_LOST
	PDU_XTRA_ERR_USB_DISCONNECT
<b>PDU_ERR_EVT_VCI_HARDWARE_FAULT</b>	
PDU_ERR_EVT_VCI_HARDWARE_FAULT	PDU_XTRA_ERR_RAM
	PDU_XTRA_ERR_FLASH
	PDU_XTRA_ERR_FILE_SYS
<b>PDU_ERR_EVT_INIT_ERROR</b>	
PDU_ERR_EVT_INIT_ERROR	PDU_XTRA_ERR_ISO_INVALID_KEYWORD
	PDU_XTRA_ERR_ISO_INIT_SEQ_ERROR

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

## Annex E (normative)

### Application defined tags

In order to facilitate usage of callback routines and to provide best performance possible, the D-PDU API makes use of an application defined void \* pointer (in the following called “tag”). Even though the application is free to use the tag or not, and what to store with the tag, the tag’s general intended function is as follows.

Usually, an application dealing with an API handle has to store additional data in conjunction with the handle. For example, the application could combine a ComPrimitive handle, retrieved from the D-PDU API, with a list of corresponding service qualifiers into a single internal data structure. The application can now internally refer to the whole information set by using a single pointer to that structure. When the D-PDU API implementation executes the event callback function provided by the application, it returns a ComPrimitive handle. Unfortunately, if no additional information is provided, the application has to iterate through a list of structures, in order to find the internal data structure with the corresponding service qualifiers (because that qualifier is not being referenced within the ComPrimitive handle).

This shortcoming has been eliminated, as the D-PDU API allows for storing a void \* pointer within a ComPrimitive, it returns the tag when executing the callback function. The application passes the pointer to the internal data structure as tag value to PDUStartComPrimitive. When the application’s event callback function is executed, the application can now directly fetch the corresponding service qualifier without iterating through a list of structures.

The client application may use the tag pointer as received from the D-PDU API, always assuming that it can rely on the value returned by the D-PDU API in a callback function or a result item. If the client application does not trust the D-PDU API, it may instead use its own list of mapped pointers to avoid the possibility of the tag values being unexpectedly modified. Tag values are not necessary since the callback function and result item always contain the corresponding handles.

Reference document:

- Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects Douglas C. Schmidt, Michael Stal
- See also <http://www.cs.wustl.edu/~schmidt/PDF/ACT.pdf>

Tags are available for the following functionality:

- pAPITag: D-PDU API Library (default system level callback);
- pCllTag: ComLogicalLink Tag (ComLogical link callback);
- pCoRTag: ComPrimitive Tag (returned in the PDU\_RESULT\_DATA for a ComLogicalLink callback).

## Annex F (normative)

### RDF and MDF description files

#### F.1 D-PDU API root description file (RDF)

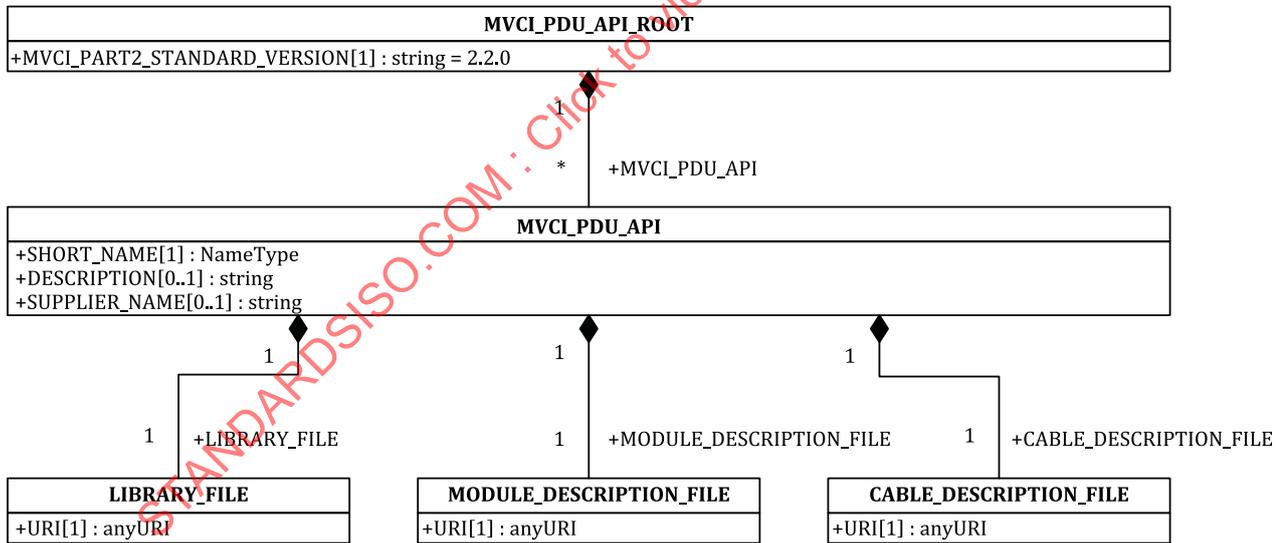
##### F.1.1 General

The D-PDU API root description file (RDF) is the central entry point for all applications accessing MVCI protocol modules. The location of this file, its structure and the procedure to get access to it, is defined for Windows and Linux platforms (see 8.7). Taking this as a basis, applications can automatically determine the available resources without any prior knowledge about which MVCI protocol modules and D-PDU API implementations are installed. Due to a standardized XML Schema structure, the application may even offer a generic user dialog to configure and operate all MVCI protocol modules, their protocols and bus types.

##### F.1.2 UML diagram of RDF

Figure F.1 shows the UML class diagram of the RDF. Every D-PDU API implementation shall add an “MVCI\_PDU\_API” element to the “MVCI\_PDU\_API\_ROOT” element. As a minimum, the “MVCI\_PDU\_API” element contains a symbolic name identifying the implementation (“SHORT\_NAME” element), a full file path onto its software library (“LIBRARY\_FILE” element), its MVCI module description file (“MODULE\_DESCRIPTION\_PATH” element) and the cable description file (“CABLE\_DESCRIPTION\_FILE” element). Optionally, this element may also include a short description and the supplier name related to the respective implementation.

**NOTE** Each DLL/library used at runtime belongs to one MVCI\_PDU\_API entry which exactly refers to one MDF and one CDF. The client application decides which MVCI\_PDU\_API entry or entries to use.



**Figure F.1 —UML class diagram of D-PDU API root description file (RDF)**

## F.2 MVCI module description file (MDF)

### F.2.1 General

The MVCI module description file (MDF) describes all ComParams, I/O controls, bus types, protocols and resources supported by a specific MVCI protocol module. In addition, if the D-PDU API implementation supports more than one type of MVCI protocol module, the MDF may list and describe all supported modules. All definitions of ComParams, bus types, protocols, etc., inside the configuration description files may be shared among the module definition in the same file.

Each element in the MDF contains a symbolic name and its corresponding numeric ID. An application could provide a user interface showing the symbolic names, e.g. for a list of ComParams. When the application needs to get or set the ComParams at the API level, it would map the symbolic names onto the corresponding IDs.

The application may also operate the D-PDU API without extracting the symbolic names and IDs at runtime. If the use case excludes frequent changes to the MDFs, simple applications may also hard-code all necessary IDs and operate the D-PDU API without parsing any MDF at runtime.

Optionally, each element in this type of the MDF may have a short description.

### F.2.2 ComParam string format

#### F.2.2.1 General

ComParam default values, min values and max values are stored as strings in the MDF file. The strings are written as a sequence of numbers separated by spaces. The numbers themselves are to be interpreted as follows.

If the first character is 0 and the second character is not “x” or “X”, the number is interpreted as an octal integer; otherwise, it is interpreted as a decimal number. If the first character is “0” and the second character is “x” or “X”, the number is interpreted as a hexadecimal integer. If the first character is “1” to “9”, the number is interpreted as a decimal integer. Any number may be preceded by a + or – sign to indicate sign.

#### F.2.2.2 String format for numeric values (data type PDU\_PT\_UNUMx or PDU\_PT\_SNUMx)

Most ComParams values are in a simple single number format like PDU\_PT\_UNUM32. The default value, min value and max value in the MDF just contain a single number for these simple types.

EXAMPLE 1 Contents for baud rate: ComParam = CP\_Baudrate: 10400.

```
<COMPARAM EID="ID1014">
  <ID>14</ID>
  <SHORT_NAME>CP_Baudrate</SHORT_NAME>
  <DESCRIPTION> Define the baud rate used for the physical vehicle bus</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>10400 </DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
```

EXAMPLE 2 Same as EXAMPLE 1 with hexadecimal number.

```
<COMPARAM EID="ID1014">
  <ID>14</ID>
  <SHORT_NAME>CP_Baudrate</SHORT_NAME>
  <DESCRIPTION> Define the baud rate used for the physical vehicle bus</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0x28A0</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
```

```
<LAYER>PHYSICAL</LAYER>
</COMPARAM>
```

### F.2.2.3 String format for bytefield values (data type PDU\_PT\_BYTEFIELD)

A bytefield structure essentially contains an array of bytes. The structure includes a maximum number of bytes, actual number of bytes and the array of bytes (see B.3.3.1).

EXAMPLE 1 Byte field contents for tester present message: ComParam = CP\_TesterPresentMessage: max bytes = 12, actual bytes = 2, data = 0x3E, 0x01.

```
<COMPARAM EID="ID1015">
  <ID>15</ID>
  <SHORT_NAME>CP_TesterPresentMessages</SHORT_NAME>
  <DESCRIPTION> Define the Tester Present Message used to keep a diagnostic session
  active on a vehicle serial bus. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_BYTEFIELD</DATA_TYPE>
  <DEFAULT_VALUE>0x0C 0x02 0x3E 0x01</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
```

EXAMPLE 2 Same as EXAMPLE 1 with a mixture of hexadecimal and decimal numbers.

```
<COMPARAM EID="ID1015">
  <ID>15</ID>
  <SHORT_NAME>CP_TesterPresentMessages</SHORT_NAME>
  <DESCRIPTION> Define the Tester Present Message used to keep a diagnostic session
  active on a vehicle serial bus. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_BYTEFIELD</DATA_TYPE>
  <DEFAULT_VALUE>12 2 0x3E 1</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
```

### F.2.2.4 String format for long field structures (data type PDU\_PT\_LONGFIELD)

A longfield structure essentially contains an array of longs. The structure includes a maximum number of longs, actual number of longs and the array of longs (see B.3.3.3).

EXAMPLE 1 Longfield contents for Can Baudrate Record: ComParam = CP\_CanBaudrateRecord: max longs = 12, actual longs = 3, data = 500000, 250000, 125000.

```
<COMPARAM EID="ID1016">
  <ID>16</ID>
  <SHORT_NAME>CP_CanBaudrateRecord</SHORT_NAME>
  <DESCRIPTION> List of baud rates to use during an OBD CAN initialization sequence.
</DESCRIPTION>
  <DATA_TYPE>PDU_PT_LONGFIELD</DATA_TYPE>
  <DEFAULT_VALUE>12 3 500000 250000 125000</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
```

EXAMPLE 2 Same as EXAMPLE 1 with a mixture of hexadecimal and decimal numbers.

```
<COMPARAM EID="ID1016">
  <ID>16</ID>
  <SHORT_NAME>CP_CanBaudrateRecord</SHORT_NAME>
  <DESCRIPTION> List of baud rates to use during an OBD CAN initialization sequence.
</DESCRIPTION>
  <DATA_TYPE>PDU_PT_LONGFIELD</DATA_TYPE>
  <DEFAULT_VALUE>0x0C 3 500000 250000 0x1e848</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
```

### F.2.2.5 String format for complex structures (data type PDU\_PT\_STRUCTFIELD)

A structfield structure essentially contains an array of specific types of structures. Each structure in the array is separated by a space similar to PDU\_PT\_BYTEFIELD and PDU\_PT\_LONGFIELD ComParam types. The structure includes a structure type, maximum number of structures, actual number of structures and the array of specific structures (see B.3.3.2).

EXAMPLE 1 Structfield contents for Can Session Timing override: ComParam = CP\_SessionTimingOverride: Structure Type = PDU\_CPST\_SESSION\_TIMING = 1, max num structures = 255, actual num structures = 2.

data = structure (1)

```
UNUM16 session = 1
UNUM8 P2Max_high = 60
UNUM8 P2Max_low = 25
UNUM8 P2Star_high = 250
UNUM8 P2Star_low = 50
```

data = structure (2)

```
UNUM16 session = 2
UNUM8 P2Max_high = 65
UNUM8 P2Max_low = 20
UNUM8 P2Star_high = 255
UNUM8 P2Star_low = 55
```

```
</COMPARAM>
<COMPARAM EID="ID1017">
  <ID>17</ID>
  <SHORT_NAME>CP_SessionTimingOverride</SHORT_NAME>
  <DESCRIPTION> Override the ECU response to a set session timing command. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_STRUCTFIELD</DATA_TYPE>
  <DEFAULT_VALUE>1 255 2 1 60 25 250 50 2 65 20 255 55 </DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
```

EXAMPLE 2 Same as EXAMPLE 1 with a mixture of hexadecimal and decimal numbers.

```
</COMPARAM>
<COMPARAM EID="ID1017">
  <ID>17</ID>
  <SHORT_NAME>CP_SessionTimingOverride</SHORT_NAME>
  <DESCRIPTION> Override the ECU response to a set session timing command. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_STRUCTFIELD</DATA_TYPE>
```

```

<DEFAULT_VALUE>1 0xFF 2 1 60 25 250 50 0x02 65 20 0xFF 55 </DEFAULT_VALUE>
<CLASS>TIMING</CLASS>
<LAYER>APPLICATION</LAYER>
</COMPARAM>
    
```

### F.2.3 ComParam resolution tag

The MDF file contains an additional string value that specifies the resolution and unit of the ComParam. This is only used for informational purposes. The resolution means that the ComParams set via PDUSetComParam and received via PDUGetComParam will be in the defined resolution units.

If the CP\_P2Star has a resolution of 0,5 ms, then the client should send 4 000 to set the ComParam to 2 s.

### F.2.4 UML diagram of MDF

Figure F.2 shows the structure of the MVCI module description file. All elements attached to the “MVCI\_MODULE\_DESCRIPTION” element may exist multiple times, referring to different contents.

Within the configuration, bus types are clearly separated from protocols because multiple protocols may run on the same bus type. The bus types are defined with the “BUSTYPE” element. Besides name, ID and description, this element has a list of ComParams. The referenced ComParams need to be defined in the same file.

Protocols are defined with the “PROTOCOL” element and contain exactly the same sub-elements as the bus type element.

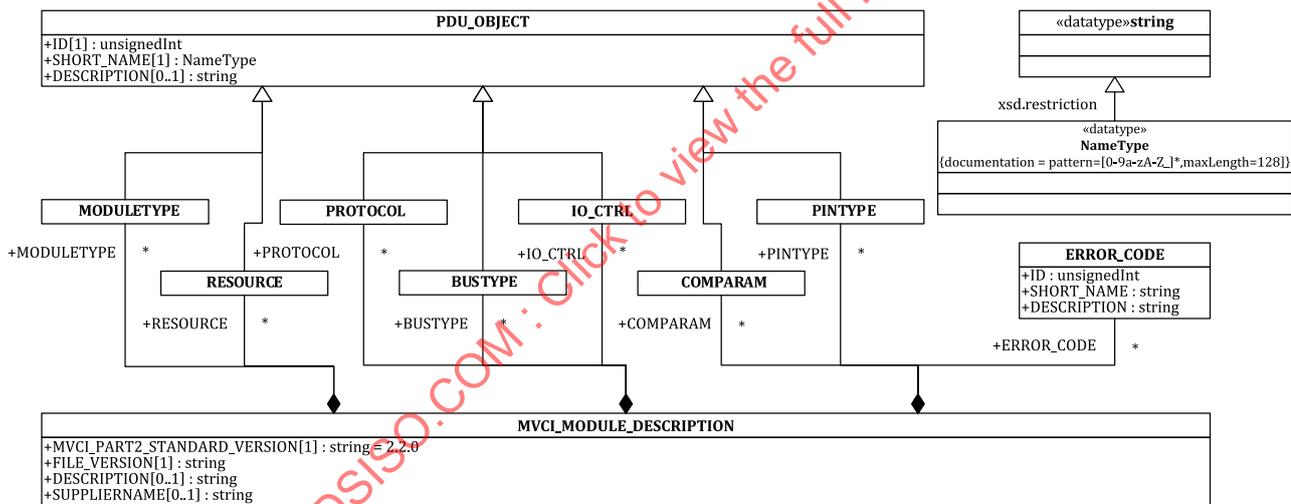


Figure F.2 — UML class diagram of MVCI module description file (MDF)

### F.2.5 UML diagram of MDF elements COMPARAM

ComParams are defined with the “COMPARAM” element and contain mandatory entries for a symbolic name (“SHORT\_NAME” element), an ID (“ID” element) and the ComParam type (“COMPARAM\_TYPE” element) (see B.3.3).

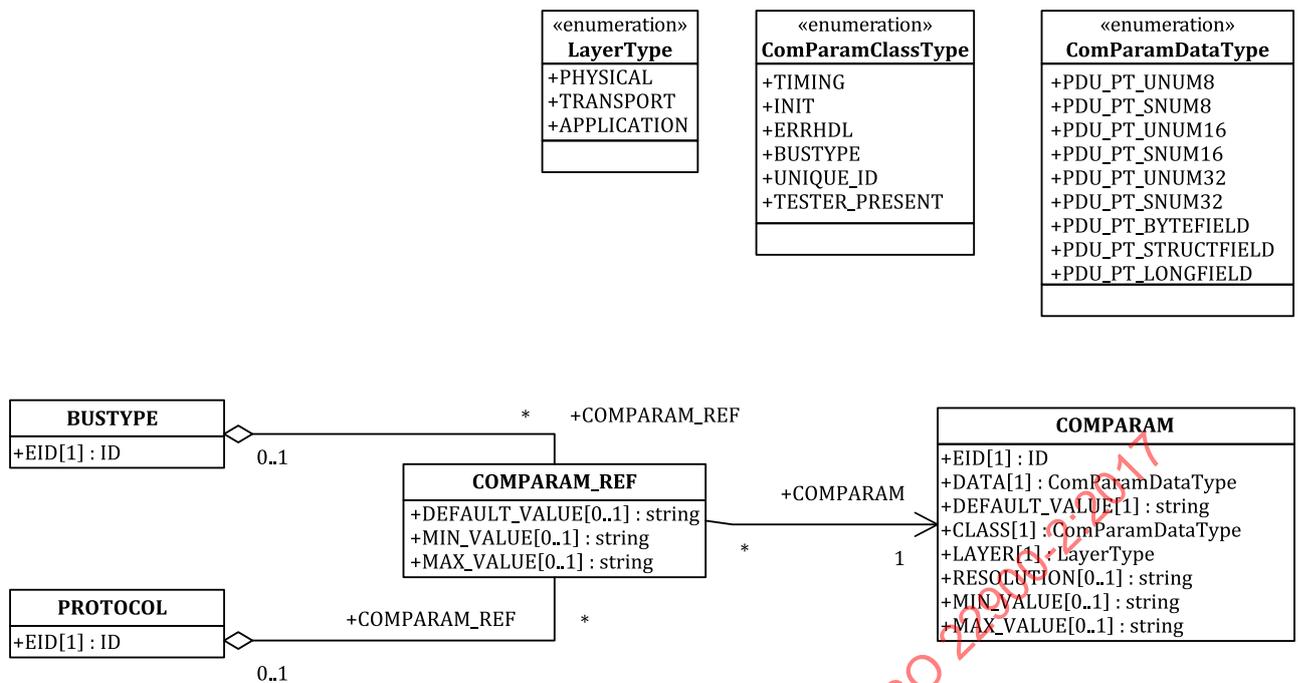


Figure F.3 — UML class diagram of element COMPARAM

### F.2.6 UML diagram of MDF element RESOURCE

Resources consist of a bus type, and one protocol supported on that bus type, and are defined by the “RESOURCE” element. Besides name, ID and description, this element has a bus type, a protocol and a list of pin numbers as sub-elements. Bus types and protocols are referenced by an ID/IDREF mechanism and shall be defined within the same file. The list of pin numbers contains all pins this resource occupies on the vehicles DLC interface. For example, for SAE J1850 VPW, the list would include one pin, whereas for CAN, it would include two pins, one for CAN High and one for CAN Low.

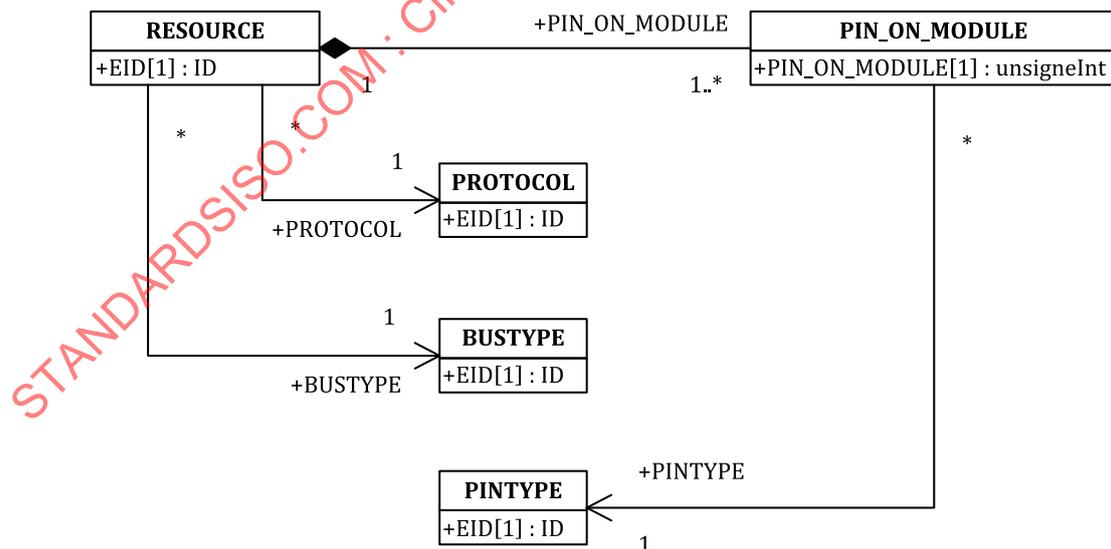
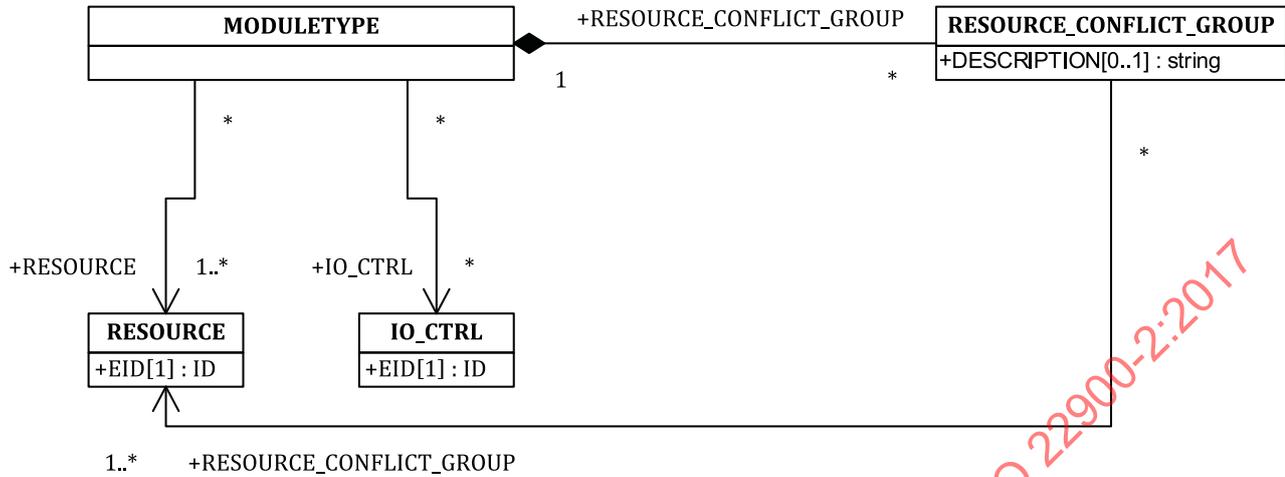


Figure F.4 — UML class diagram of element RESOURCE

**F.2.7 UML diagram of MDF element MODULETYPE**

As the last element in the logical chain, the modules are defined by the “MODULETYPE” element and have lists of I/O controls and resources, besides name, ID and description. I/O controls are specified using the “IO\_CTRL” element. The elements are defined by name, ID and description. A detailed view on I/O controls is depicted in 8.5.



**Figure F.5 — UML class diagram of element MODULETYPE, RESOURCE and IO\_CTRL**

**F.3 Cable description file (CDF)**

**F.3.1 General**

The cable description file (depicted in Figure F.6) closes the gap between the cable’s pin assignment and the vendor-specific pin assignment on the MVCI protocol module’s DLC interface.

**F.3.2 UML diagram of CDF**

Each cable has its own “CABLE” element and the mandatory sub-elements “SHORT\_NAME”, “DLC\_TYPE” and “ID”. The sub-element “DESCRIPTION” is again optional. Besides those sub-elements, there are two additional mandatory sub-elements: a “MAPPING” sub-element for each resource mapped and one “CABLE\_IDENTIFICATION” sub-element.

The “MAPPING” sub-element includes two mandatory sub-elements. The “PIN\_ON\_DLC” sub-element defines the pin number on the DLC, while the “PIN\_ON\_VCI” sub-element defines the corresponding pin number on the MVCI protocol module.

The “CABLE\_IDENTIFICATION” sub-element contains the “CABLE\_ID” sub-element, and optionally the “CABLE\_ID\_PIN” sub-element, which contains a resistance value and a pin number for automatic cable identification.

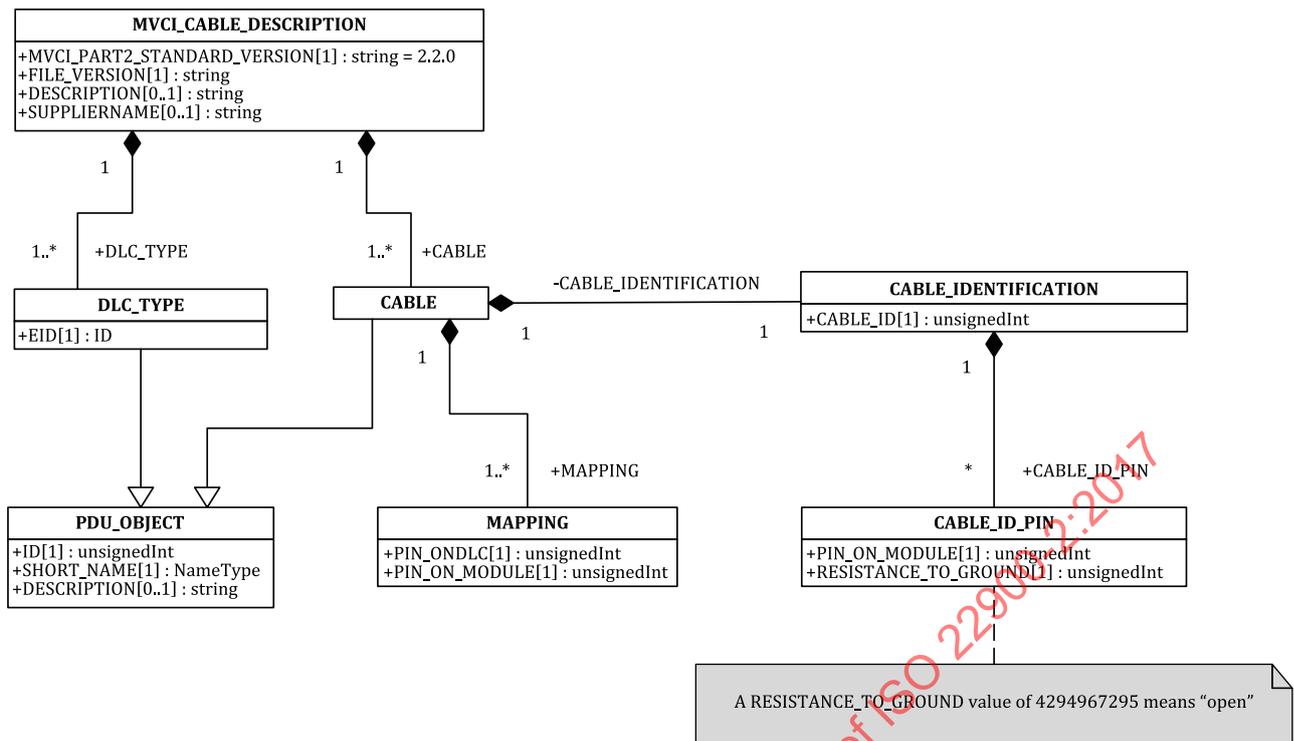


Figure F.6 — UML class diagram of cable description file (CDF)

## F.4 XML schema

XML schema for the D-PDU API Description Files.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="BUSTYPE">
    <xsd:complexContent>
      <xsd:extension base="PDU_OBJECT">
        <xsd:sequence>
          <xsd:element name="COMPARAM_REF" minOccurs="0" maxOccurs="unbounded"
type="COMPARAM_REF"/>
        </xsd:sequence>
        <xsd:attribute name="EID" use="required" type="xsd:ID"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="CABLE">
    <xsd:complexContent>
      <xsd:extension base="PDU_OBJECT">
        <xsd:sequence>
          <xsd:element name="CABLE_IDENTIFICATION" minOccurs="1" maxOccurs="1"
type="CABLE_IDENTIFICATION"/>
          <xsd:element name="MAPPING" minOccurs="1" maxOccurs="unbounded"
type="MAPPING"/>
          <xsd:element name="DLCTYPE" minOccurs="1" maxOccurs="1">
            <xsd:complexType>
              <xsd:attribute name="IDREF" type="xsd:IDREF" />
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

```

```

</xsd:complexType>
<xsd:complexType name="CABLE_DESCRIPTION_FILE">
  <xsd:attribute name="URI" use="required" type="xsd:anyURI"/>
</xsd:complexType>
<xsd:complexType name="CABLE_ID_PIN">
  <xsd:sequence>
    <xsd:element name="PIN_ON_MODULE" minOccurs="1" maxOccurs="1"
type="xsd:unsignedInt"/>
    <xsd:element name="RESISTANCE_TO_GROUND" minOccurs="1" maxOccurs="1"
type="xsd:unsignedInt"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CABLE_IDENTIFICATION">
  <xsd:sequence>
    <xsd:element name="CABLE_ID" minOccurs="1" maxOccurs="1" type="xsd:unsignedInt"/>
    <xsd:element name="CABLE_ID_PIN" minOccurs="0" maxOccurs="unbounded"
type="CABLE_ID_PIN"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="COMPARAM">
  <xsd:complexContent>
    <xsd:extension base="PDU_OBJECT">
      <xsd:sequence>
        <xsd:element name="DATA_TYPE" minOccurs="1" maxOccurs="1"
type="ComParamDataType"/>
        <xsd:element name="DEFAULT_VALUE" minOccurs="1" maxOccurs="1"
type="xsd:string"/>
        <xsd:element name="CLASS" minOccurs="1" maxOccurs="1"
type="ComParamClassType"/>
        <xsd:element name="LAYER" minOccurs="1" maxOccurs="1" type="LayerType"/>
        <xsd:element name="RESOLUTION" minOccurs="0" maxOccurs="1"
type="xsd:string"/>
        <xsd:element name="MIN_VALUE" minOccurs="0" maxOccurs="1" type="xsd:string"/>
        <xsd:element name="MAX_VALUE" minOccurs="0" maxOccurs="1" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="EID" use="required" type="xsd:ID"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="COMPARAM_REF">
  <xsd:sequence>
    <xsd:element name="DEFAULT_VALUE" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="MIN_VALUE" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="MAX_VALUE" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="COMPARAM" minOccurs="1" maxOccurs="1">
      <xsd:complexType>
        <xsd:attribute name="IDREF" type="xsd:IDREF" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="ComParamClassType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="undefined"/>
    <xsd:enumeration value="TIMING"/>
    <xsd:enumeration value="INIT"/>
    <xsd:enumeration value="COM"/>
    <xsd:enumeration value="ERRHDL"/>
  </xsd:restriction>
</xsd:simpleType>

```



```

        <xsd:enumeration value="BUSTYPE"/>
        <xsd:enumeration value="UNIQUE_ID"/>
        <xsd:enumeration value="TESTER_PRESENT"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ComParamDataType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="undefined"/>
        <xsd:enumeration value="PDU_PT_UNUM8"/>
        <xsd:enumeration value="PDU_PT_SNUM8"/>
        <xsd:enumeration value="PDU_PT_UNUM16"/>
        <xsd:enumeration value="PDU_PT_SNUM16"/>
        <xsd:enumeration value="PDU_PT_UNUM32"/>
        <xsd:enumeration value="PDU_PT_SNUM32"/>
        <xsd:enumeration value="PDU_PT_BYTEFIELD"/>
        <xsd:enumeration value="PDU_PT_STRUCTFIELD"/>
        <xsd:enumeration value="PDU_PT_LONGFIELD"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="DLCTYPE">
    <xsd:complexContent>
        <xsd:extension base="PDU_OBJECT">
            <xsd:attribute name="EID" use="required" type="xsd:ID"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ERROR_CODE">
    <xsd:sequence>
        <xsd:element name="ID" minOccurs="1" maxOccurs="1" type="xsd:unsignedInt"/>
        <xsd:element name="SHORT_NAME" minOccurs="1" maxOccurs="1" type="NameType"/>
        <xsd:element name="DESCRIPTION" minOccurs="1" maxOccurs="1" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IO_CTRL">
    <xsd:complexContent>
        <xsd:extension base="PDU_OBJECT">
            <xsd:attribute name="EID" use="required" type="xsd:ID"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="LayerType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="undefined"/>
        <xsd:enumeration value="PHYSICAL"/>
        <xsd:enumeration value="TRANSPORT"/>
        <xsd:enumeration value="APPLICATION"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="LIBRARY_FILE">
    <xsd:attribute name="URI" use="required" type="xsd:anyURI"/>
</xsd:complexType>
<xsd:complexType name="MAPPING">
    <xsd:sequence>
        <xsd:element name="PIN_ON_DLC" minOccurs="1" maxOccurs="1" type="xsd:unsignedInt"/>
        <xsd:element name="PIN_ON_MODULE" minOccurs="1" maxOccurs="1" type="xsd:unsignedInt"/>
    </xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="MODULE_DESCRIPTION_FILE">
  <xsd:attribute name="URI" use="required" type="xsd:anyURI"/>
</xsd:complexType>
<xsd:complexType name="MODULETYPE">
  <xsd:complexContent>
    <xsd:extension base="PDU_OBJECT">
      <xsd:sequence>
        <xsd:element name="RESOURCE_CONFLICT_GROUP" minOccurs="0"
maxOccurs="unbounded" type="RESOURCE_CONFLICT_GROUP"/>
        <xsd:element name="IO_CTRL" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:attribute name="IDREF" type="xsd:IDREF" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="RESOURCE" minOccurs="1" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:attribute name="IDREF" type="xsd:IDREF" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MVCI_CABLE_DESCRIPTION">
  <xsd:sequence>
    <xsd:element name="DESCRIPTION" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="SUPPLIER_NAME" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="DLCTYPE" minOccurs="1" maxOccurs="unbounded" type="DLCTYPE"/>
    <xsd:element name="CABLE" minOccurs="1" maxOccurs="unbounded" type="CABLE"/>
  </xsd:sequence>
  <xsd:attribute name="MVCI_PART2_STANDARD_VERSION" use="required" type="xsd:string"
fixed="2.2.1"/>
  <xsd:attribute name="FILE_VERSION" use="required" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="MVCI_CABLE_DESCRIPTION" type="MVCI_CABLE_DESCRIPTION"/>
<xsd:complexType name="MVCI_MODULE_DESCRIPTION">
  <xsd:sequence>
    <xsd:element name="DESCRIPTION" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="SUPPLIER_NAME" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="PINTYPE" minOccurs="0" maxOccurs="unbounded" type="PINTYPE"/>
    <xsd:element name="MODULETYPE" minOccurs="0" maxOccurs="unbounded"
type="MODULETYPE"/>
    <xsd:element name="RESOURCE" minOccurs="0" maxOccurs="unbounded" type="RESOURCE"/>
    <xsd:element name="PROTOCOL" minOccurs="0" maxOccurs="unbounded" type="PROTOCOL"/>
    <xsd:element name="BUSTYPE" minOccurs="0" maxOccurs="unbounded" type="BUSTYPE"/>
    <xsd:element name="IO_CTRL" minOccurs="0" maxOccurs="unbounded" type="IO_CTRL"/>
    <xsd:element name="COMPARAM" minOccurs="0" maxOccurs="unbounded" type="COMPARAM"/>
    <xsd:element name="ERROR_CODE" minOccurs="0" maxOccurs="unbounded"
type="ERROR_CODE"/>
  </xsd:sequence>
  <xsd:attribute name="MVCI_PART2_STANDARD_VERSION" use="required" type="xsd:string"
fixed="2.2.1"/>
  <xsd:attribute name="FILE_VERSION" use="required" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="MVCI_MODULE_DESCRIPTION" type="MVCI_MODULE_DESCRIPTION"/>
<xsd:complexType name="MVCI_PDU_API">
  <xsd:sequence>
    <xsd:element name="SHORT_NAME" minOccurs="1" maxOccurs="1" type="NameType"/>
    <xsd:element name="DESCRIPTION" minOccurs="0" maxOccurs="1" type="xsd:string"/>

```

```

        <xsd:element name="SUPPLIER_NAME" minOccurs="0" maxOccurs="1" type="xsd:string"/>
        <xsd:element name="LIBRARY_FILE" minOccurs="1" maxOccurs="1" type="LIBRARY_FILE"/>
        <xsd:element name="MODULE_DESCRIPTION_FILE" minOccurs="1" maxOccurs="1"
type="MODULE_DESCRIPTION_FILE"/>
        <xsd:element name="CABLE_DESCRIPTION_FILE" minOccurs="1" maxOccurs="1"
type="CABLE_DESCRIPTION_FILE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MVCI_PDU_API_ROOT">
    <xsd:sequence>
        <xsd:element name="MVCI_PDU_API" minOccurs="0" maxOccurs="unbounded"
type="MVCI_PDU_API"/>
    </xsd:sequence>
    <xsd:attribute name="MVCI_PART2_STANDARD_VERSION" use="required" type="xsd:string"
fixed="2.2.1"/>
</xsd:complexType>
<xsd:element name="MVCI_PDU_API_ROOT" type="MVCI_PDU_API_ROOT"/>
<xsd:simpleType name="NameType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9a-zA-Z_]*"/>
        <xsd:maxLength value="128"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="PDU_OBJECT">
    <xsd:sequence>
        <xsd:element name="ID" minOccurs="1" maxOccurs="1" type="xsd:unsignedInt"/>
        <xsd:element name="SHORT_NAME" minOccurs="1" maxOccurs="1" type="NameType"/>
        <xsd:element name="DESCRIPTION" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PIN_ON_MODULE">
    <xsd:sequence>
        <xsd:element name="PIN_ON_MODULE" minOccurs="1" maxOccurs="1"
type="xsd:unsignedInt"/>
        <xsd:element name="PINTYPE" minOccurs="1" maxOccurs="1">
            <xsd:complexType>
                <xsd:attribute name="IDREF" type="xsd:IDREF" />
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PINTYPE">
    <xsd:complexContent>
        <xsd:extension base="PDU_OBJECT">
            <xsd:attribute name="EID" use="required" type="xsd:ID"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PROTOCOL">
    <xsd:complexContent>
        <xsd:extension base="PDU_OBJECT">
            <xsd:sequence>
                <xsd:element name="COMPARAM_REF" minOccurs="0" maxOccurs="unbounded"
type="COMPARAM_REF"/>
            </xsd:sequence>
            <xsd:attribute name="EID" use="required" type="xsd:ID"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

```

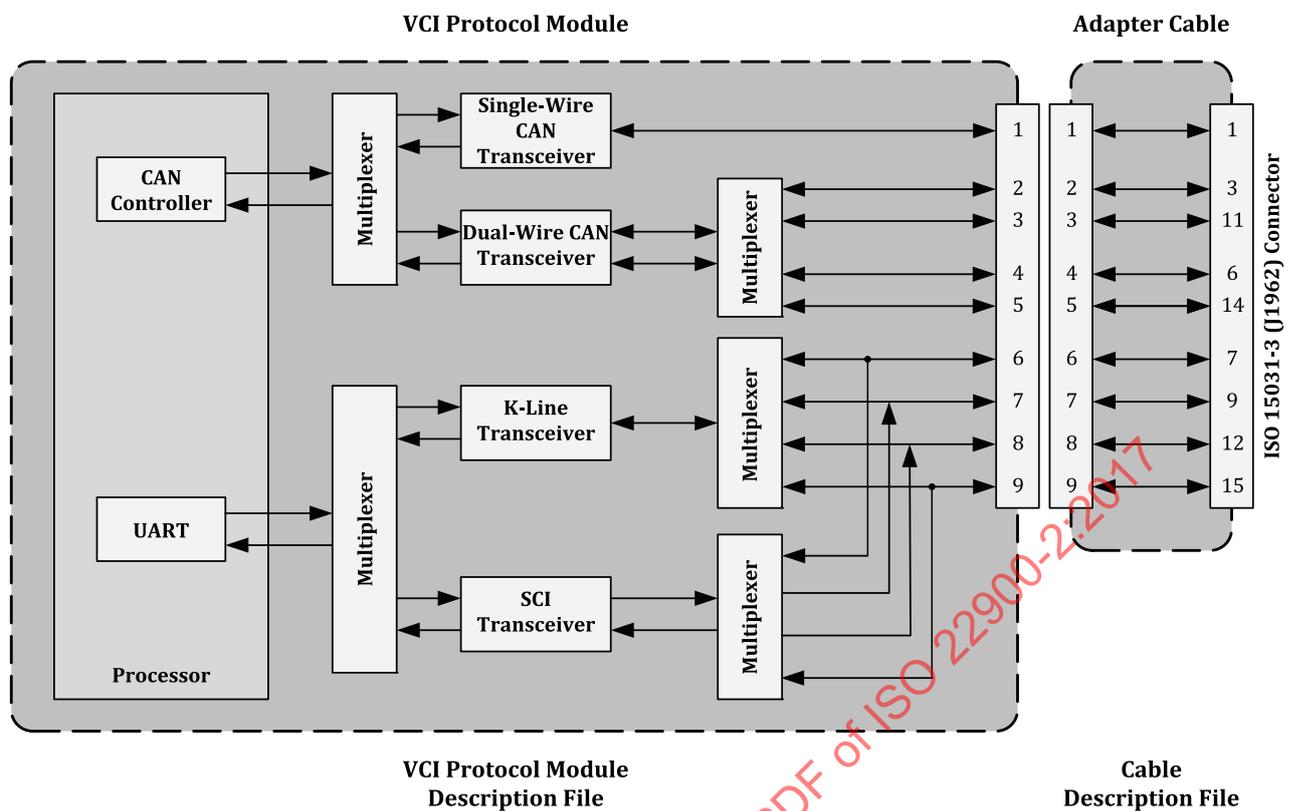
<xsd:complexType name="RESOURCE">
  <xsd:complexContent>
    <xsd:extension base="PDU_OBJECT">
      <xsd:sequence>
        <xsd:element name="PIN_ON_MODULE" minOccurs="1" maxOccurs="unbounded"
type="PIN_ON_MODULE"/>
        <xsd:element name="BUSTYPE" minOccurs="1" maxOccurs="1">
          <xsd:complexType>
            <xsd:attribute name="IDREF" type="xsd:IDREF" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="PROTOCOL" minOccurs="1" maxOccurs="1">
          <xsd:complexType>
            <xsd:attribute name="IDREF" type="xsd:IDREF" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="EID" use="required" type="xsd:ID"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="RESOURCE_CONFLICT_GROUP">
  <xsd:sequence>
    <xsd:element name="DESCRIPTION" minOccurs="0" maxOccurs="1" type="xsd:string"/>
    <xsd:element name="RESOURCE" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="IDREF" type="xsd:IDREF" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

**F.5 Description file examples**

**F.5.1 Example MVCI protocol module**

This subclause will demonstrate how a D-PDU API's and MVCI protocol module's capabilities have to be described according to the D-PDU API XML schema. The capabilities of an example MVCI protocol module are shown in Figure F.7. The example is considering different data link controllers, physical link transceivers and diagnostic connectors. Also, multiplexing and shared pin resources are considered. All combinations of protocols, data link layers and physical link layers supported by this example are listed below the figure. The pin numbers listed refer to the pin assignment on the cable, not on the example MVCI protocol module.

**Supported resources**

Dual-wire CAN on pins 3 and 11 (Protocols: ISO 11898, ISO 15765, SAE J1939)

Dual-wire CAN on pins 6 and 14 (Protocols: ISO11898, ISO15765, SAE J1939)

Single-wire CAN on pin 1 (Protocols: ISO 11898, ISO 15765)

ISO K-Line on pin 7 (Protocol: KWP2000, ISO 9141-2)

ISO K-Line on pin 9 (Protocol: KWP2000, ISO 9141-2)

ISO K-Line on pin 12 (Protocol: KWP2000, ISO 9141-2)

ISO K-Line on pin 15 (Protocol: KWP2000, ISO 9141-2)

SCI\_B\_ENGINE - Tx on pin 12 and Rx on pin 7 (Protocol: SAE J2610)

SCI\_B\_TRANS - Tx on pin 9 and Rx on pin 15 (Protocol: SAE J2610)

**Figure F.7 — Example MSCI protocol module**

The example in Figure 7 supports the following resources:

- dual-wire CAN on pins 3 and 11 (Protocols: ISO 11898, ISO 15765, SAE J1939);
- dual-wire CAN on pins 6 and 14 (Protocols: ISO 11898, ISO 15765, SAE J1939);
- single-wire CAN on pin 1 (Protocols: ISO 11898, ISO 15765);
- ISO K-Line on pin 7 (Protocol: KWP2000, ISO 9141-2);
- ISO K-Line on pin 9 (Protocol: KWP2000, ISO 9141-2);
- ISO K-Line on pin 12 (Protocol: KWP2000, ISO 9141-2);
- ISO K-Line on pin 15 (Protocol: KWP2000, ISO 9141-2);
- SCI\_B\_ENGINE - Tx on pin 12 and Rx on pin 7 (Protocol: SAE J2610);
- SCI\_B\_TRANS - Tx on pin 9 and Rx on pin 15 (Protocol: SAE J2610).

## F.5.2 Example root description file

According to the MVCI protocol module setup (see example root description file), there shall be at least two D-PDU API entries in the root description file. The XML entry MVCI\_PDU\_API "D\_PDU\_API\_1" is the implementation supporting this example MVCI protocol module. Using XML entry LIBRARY\_FILE (contains filename including full path), the application will find the D-PDU API implementation's dynamic library at "C:\tmp1\D-PDU API Example.dll". Similarly, the application may locate the D-PDU API implementation's MDF and CDF file by reading the XML entry MODULE\_DESCRIPTION\_FILE and CABLE\_DESCRIPTION\_FILE (both contain filename including full path).

EXAMPLE           Root description file (RDF file).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- D-PDU-API root file -->
<MVCI_PDU_API_ROOT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Data\Dev\eclipse\workspace\PduApi\pdu.xsd"
MVCI_PART2_STANDARD_VERSION="2.2.1">
  <MVCI_PDU_API>
    <SHORT_NAME>D_PDU_API_1</SHORT_NAME>
    <DESCRIPTION>D-PDU-API Implementation used for MVCI of vendor #1</DESCRIPTION>
    <SUPPLIER_NAME>Vehicle Doctor Ltd.</SUPPLIER_NAME>
    <LIBRARY_FILE URI="file:/c:/tmp1/PDUAPI_VEHICLEDOCTOR_2.2.1.dll"/>
    <MODULE_DESCRIPTION_FILE URI="file:/c:/tmp1/MDF_VEHICLEDOCTOR_VCI1_2.2.1.xml"/>
    <CABLE_DESCRIPTION_FILE URI="file:/c:/tmp1/CDF_VEHICLEDOCTOR_VCI1_2.2.1.xml"/>
  </MVCI_PDU_API>
  <MVCI_PDU_API>
    <SHORT_NAME>D_PDU_API_2</SHORT_NAME>
    <DESCRIPTION>D-PDU-API Implementation used for MVCI of vendor #2</DESCRIPTION>
    <SUPPLIER_NAME>OBD Bob Ltd.</SUPPLIER_NAME>
    <LIBRARY_FILE URI="file:/c:/tmp2/PDUAPI_OBDBOB_2.2.1.dll"/>
    <MODULE_DESCRIPTION_FILE URI="file:/c:/tmp2/MDF_OBDBOB_VCIx_2.2.1.xml"/>
    <CABLE_DESCRIPTION_FILE URI="file:/c:/tmp2/CDF_OBDBOB_VCIx_2.2.1.xml"/>
  </MVCI_PDU_API>
</MVCI_PDU_API_ROOT>
```

### F.5.3 Example module description file

The module description file (see example module description file) describes the example MVCI protocol module's capabilities in detail up to the pins seen on the MVCI protocol module itself. It does not include any mapping onto pins on a diagnostic link connector (DLC).

**EXAMPLE** Module Description file (MDF file). This is only an example and not a template based on the latest ComParam definitions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- D-PDU-API module description file -->
<MVCI_MODULE_DESCRIPTION xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="C:\Data\Dev\eclipse\workspace\PduApi\pdu.xsd" FILE_VERSION="0.0.1"
MVCI_PART2_STANDARD_VERSION="2.2.1">
  <DESCRIPTION>This is an example for a module description file</DESCRIPTION>
  <SUPPLIER_NAME>Vehicle Doctor Ltd.</SUPPLIER_NAME>
  <PINTYPE EID="ID2297">
    <ID>2000</ID>
    <SHORT_NAME>HI</SHORT_NAME>
    <DESCRIPTION>High signal</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2298">
    <ID>2001</ID>
    <SHORT_NAME>LO</SHORT_NAME>
    <DESCRIPTION>Low signal</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2299">
    <ID>2002</ID>
    <SHORT_NAME>K</SHORT_NAME>
    <DESCRIPTION>K line</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2300">
    <ID>2003</ID>
    <SHORT_NAME>L</SHORT_NAME>
    <DESCRIPTION>L line</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2301">
    <ID>2004</ID>
    <SHORT_NAME>TX</SHORT_NAME>
    <DESCRIPTION>TX lines</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2302">
    <ID>2005</ID>
    <SHORT_NAME>RX</SHORT_NAME>
    <DESCRIPTION>RX line</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2303">
    <ID>2006</ID>
    <SHORT_NAME>PLUS</SHORT_NAME>
    <DESCRIPTION>Plus line</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2304">
    <ID>2007</ID>
    <SHORT_NAME>MINUS</SHORT_NAME>
    <DESCRIPTION>Minus line</DESCRIPTION>
  </PINTYPE>
  <PINTYPE EID="ID2305">
    <ID>2008</ID>
```

```

    <SHORT_NAME>SINGLE</SHORT_NAME>
    <DESCRIPTION>Single line</DESCRIPTION>
  </PINTYPE>
<PINTYPE EID="ID2306">
  <ID>2009</ID>
  <SHORT_NAME>IGN</SHORT_NAME>
  <DESCRIPTION>Pin for detection of Ignition Sense State</DESCRIPTION>
</PINTYPE>
<PINTYPE EID="ID2307">
  <ID>2010</ID>
  <SHORT_NAME>PROGV</SHORT_NAME>
  <DESCRIPTION> Pin for Reading and Setting the Programmable Voltage </DESCRIPTION>
</PINTYPE>
<MODULETYPE>
  <ID>500</ID>
  <SHORT_NAME>Example_VCI_1</SHORT_NAME>
  <DESCRIPTION>Example MCCI #1 of vendor #1</DESCRIPTION>
  <RESOURCE-CONFLICT-GROUP>
    <DESCRIPTION>Conflicts with CAN controller</DESCRIPTION>
    <RESOURCE IDREF="ID11"/>
    <RESOURCE IDREF="ID12"/>
    <RESOURCE IDREF="ID13"/>
    <RESOURCE IDREF="ID14"/>
    <RESOURCE IDREF="ID15"/>
    <RESOURCE IDREF="ID16"/>
    <RESOURCE IDREF="ID17"/>
    <RESOURCE IDREF="ID18"/>
  </RESOURCE-CONFLICT-GROUP>
  <RESOURCE-CONFLICT-GROUP>
    <DESCRIPTION>Conflicts with UART</DESCRIPTION>
    <RESOURCE IDREF="ID19"/>
    <RESOURCE IDREF="ID20"/>
    <RESOURCE IDREF="ID21"/>
    <RESOURCE IDREF="ID22"/>
    <RESOURCE IDREF="ID23"/>
    <RESOURCE IDREF="ID24"/>
  </RESOURCE-CONFLICT-GROUP>
  <IO_CTRL IDREF="ID2199"/>
  <IO_CTRL IDREF="ID2200"/>
  <RESOURCE IDREF="ID11"/>
  <RESOURCE IDREF="ID12"/>
  <RESOURCE IDREF="ID13"/>
  <RESOURCE IDREF="ID14"/>
  <RESOURCE IDREF="ID15"/>
  <RESOURCE IDREF="ID16"/>
  <RESOURCE IDREF="ID17"/>
  <RESOURCE IDREF="ID18"/>
  <RESOURCE IDREF="ID19"/>
  <RESOURCE IDREF="ID20"/>
  <RESOURCE IDREF="ID21"/>
  <RESOURCE IDREF="ID22"/>
  <RESOURCE IDREF="ID23"/>
  <RESOURCE IDREF="ID24"/>
</MODULETYPE>
<MODULETYPE>
  <ID>501</ID>
  <SHORT_NAME>Example_VCI_2</SHORT_NAME>

```

```

<DESCRIPTION>Example MSCI #2 of vendor #1</DESCRIPTION>
<RESOURCE IDREF="ID23"/>
</MODULETYPE>
<RESOURCE EID="ID11">
  <ID>400</ID>
  <SHORT_NAME>DW_CAN_1</SHORT_NAME>
  <DESCRIPTION>Dual Wire CAN on pins 2 and 3 for protocol #1</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>2</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2297"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>3</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2298"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID4"/>
  <PROTOCOL IDREF="ID52"/>
</RESOURCE>
<RESOURCE EID="ID12">
  <ID>401</ID>
  <SHORT_NAME>DW_CAN_2</SHORT_NAME>
  <DESCRIPTION>Dual Wire CAN on pins 2 and 3 for protocol #2</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>2</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2297"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>3</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2298"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID4"/>
  <PROTOCOL IDREF="ID53"/>
</RESOURCE>
<RESOURCE EID="ID13">
  <ID>402</ID>
  <SHORT_NAME>DW_CAN_3</SHORT_NAME>
  <DESCRIPTION>Dual Wire CAN on pins 2 and 3 for protocol #3</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>2</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2297"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>3</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2298"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID4"/>
  <PROTOCOL IDREF="ID54"/>
</RESOURCE>
<RESOURCE EID="ID14">
  <ID>403</ID>
  <SHORT_NAME>DW_CAN_4</SHORT_NAME>
  <DESCRIPTION>Dual Wire CAN on pins 4 and 5 for protocol #1</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>4</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2297"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>

```

```

    <PIN_ON_MODULE>5</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2298"/>
</PIN_ON_MODULE>
<BUSTYPE IDREF="ID4"/>
<PROTOCOL IDREF="ID52"/>
</RESOURCE>
<RESOURCE EID="ID15">
  <ID>404</ID>
  <SHORT_NAME>DW_CAN_5</SHORT_NAME>
  <DESCRIPTION>Dual Wire CAN on pins 4 and 5 for protocol #2</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>4</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2297"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>5</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2298"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID4"/>
  <PROTOCOL IDREF="ID53"/>
</RESOURCE>
<RESOURCE EID="ID16">
  <ID>405</ID>
  <SHORT_NAME>DW_CAN_6</SHORT_NAME>
  <DESCRIPTION>Dual Wire CAN on pins 4 and 5 for protocol #3</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>4</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2297"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>5</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2298"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID4"/>
  <PROTOCOL IDREF="ID54"/>
</RESOURCE>
<RESOURCE EID="ID17">
  <ID>406</ID>
  <SHORT_NAME>SW_CAN_1</SHORT_NAME>
  <DESCRIPTION>Single Wire CAN on pin 2 for protocol #1</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>1</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2305"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID5"/>
  <PROTOCOL IDREF="ID52"/>
</RESOURCE>
<RESOURCE EID="ID18">
  <ID>407</ID>
  <SHORT_NAME>SW_CAN_2</SHORT_NAME>
  <DESCRIPTION>Single Wire CAN on pin 2 for protocol #2</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>1</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2305"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID5"/>
  <PROTOCOL IDREF="ID53"/>

```

```

</RESOURCE>
<RESOURCE EID="ID19">
  <ID>408</ID>
  <SHORT_NAME>K_LINE_1</SHORT_NAME>
  <DESCRIPTION>K-Line on pin 6</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>6</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2299"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID3"/>
  <PROTOCOL IDREF="ID50"/>
</RESOURCE>
<RESOURCE EID="ID20">
  <ID>409</ID>
  <SHORT_NAME>K_LINE_2</SHORT_NAME>
  <DESCRIPTION>K-Line on pin 7</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>7</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2299"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID3"/>
  <PROTOCOL IDREF="ID50"/>
</RESOURCE>
<RESOURCE EID="ID21">
  <ID>410</ID>
  <SHORT_NAME>K_LINE_3</SHORT_NAME>
  <DESCRIPTION>K-Line on pin 8</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>8</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2299"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID3"/>
  <PROTOCOL IDREF="ID50"/>
</RESOURCE>
<RESOURCE EID="ID22">
  <ID>411</ID>
  <SHORT_NAME>K_LINE_4</SHORT_NAME>
  <DESCRIPTION>K-Line on pin 9</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>9</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2299"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID3"/>
  <PROTOCOL IDREF="ID50"/>
</RESOURCE>
<RESOURCE EID="ID23">
  <ID>412</ID>
  <SHORT_NAME>SCI_1</SHORT_NAME>
  <DESCRIPTION>SCI on pins 6 and 8</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>6</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2302"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>8</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2301"/>
  </PIN_ON_MODULE>

```

```

<BUSTYPE IDREF="ID6"/>
<PROTOCOL IDREF="ID51"/>
</RESOURCE>
<RESOURCE EID="ID24">
  <ID>413</ID>
  <SHORT_NAME>SCI_2</SHORT_NAME>
  <DESCRIPTION>SCI on pins 7 and 9</DESCRIPTION>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>9</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2302"/>
  </PIN_ON_MODULE>
  <PIN_ON_MODULE>
    <PIN_ON_MODULE>8</PIN_ON_MODULE>
    <PINTYPE IDREF="ID2301"/>
  </PIN_ON_MODULE>
  <BUSTYPE IDREF="ID6"/>
  <PROTOCOL IDREF="ID51"/>
</RESOURCE>
<PROTOCOL EID="ID50">
  <ID>300</ID>
  <SHORT_NAME>ISO_14230_3_on_ISO_14230_2</SHORT_NAME>
  <DESCRIPTION>Keyword protocol 2000 on K-Line (ISO 14230)</DESCRIPTION>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1002"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1005"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1006"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>50</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1008"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>110</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1013"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1016"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>3</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1017"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1018"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>

```

```

    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1019"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>3</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1020"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1021"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>50000</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1022"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1023"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1024"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1025"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1026"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1027"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1028"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1031"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>51</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1100"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1101"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1102"/>
</COMPARAM_REF>
<COMPARAM_REF>

```

```

    <DEFAULT_VALUE>16</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1135"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1136"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1137"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>51</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1138"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>241</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1139"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>4</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1141"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>2</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1142"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1146"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1147"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1148"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>10</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1149"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1150"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1151"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>16</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1152"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1153"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>

```

```

    <COMPARAM IDREF="ID1154"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>241</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1163"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>600</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1164"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>50</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1165"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1166"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>600</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1167"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>120</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1168"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1169"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>10</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1170"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1171"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1172"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1173"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>50</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1174"/>
  </COMPARAM_REF>
</PROTOCOL>
<PROTOCOL EID="ID53">
  <ID>304</ID>
  <SHORT_NAME>ISO_15765_3_on_ISO_15765_2</SHORT_NAME>
  <DESCRIPTION>UDS on CAN</DESCRIPTION>
  <COMPARAM_REF>

```

```

    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1002"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1001"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1005"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>200</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1006"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1008"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1009"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1011"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1014"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>240</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1016"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>3</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1017"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>10</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1018"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1019"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>3</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1020"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1021"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>50000</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1022"/>

```

```

</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1023"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1024"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1025"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1026"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1027"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1028"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1031"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1034"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1103"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1105"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1107"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>65535</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1109"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>20</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1110"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1112"/>
</COMPARAM_REF>

```

```
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1114"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>85</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1115"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1116"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1117"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>255</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1121"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1118"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>4</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1119"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2015</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1120"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1125"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>4</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1126"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2024</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1127"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1129"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1128"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1512</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1130"/>
</COMPARAM_REF>
<COMPARAM_REF>
```

```

    <DEFAULT_VALUE>2000</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1131"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>20</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1133"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1153"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1154"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1155"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>65535</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1157"/>
</COMPARAM_REF>
</PROTOCOL>
<SHORT_NAME>ISO_14229_3_on_ISO_15765_2</SHORT_NAME>
<DESCRIPTION>UDS on CAN</DESCRIPTION>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1002"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1001"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1005"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>200</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1006"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1008"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1009"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1011"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1014"/>

```

```

</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>240</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1016"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>3</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1017"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>10</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1018"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1019"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>3</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1020"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1021"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>50000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1022"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1023"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1024"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1025"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1026"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1027"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1028"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1031"/>
</COMPARAM_REF>

```

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

```

<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1034"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1103"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1105"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1107"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>65535</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1109"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>20</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1110"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>2000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1112"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1114"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>85</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1115"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1116"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1117"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>255</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1121"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1118"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>4</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1119"/>
</COMPARAM_REF>
<COMPARAM_REF>

```

```

    <DEFAULT_VALUE>2015</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1120"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1125"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>4</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1126"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>2024</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1127"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1129"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1128"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1512</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1130"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>2000</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1131"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>20</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1133"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1153"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1154"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1155"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>65535</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1157"/>
</COMPARAM_REF>
</PROTOCOL>
<PROTOCOL EID="ID52">
    <ID>302</ID>
    <SHORT_NAME>ISO_11898_RAW</SHORT_NAME>
    <DESCRIPTION>CAN raw</DESCRIPTION>

```

```

<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1002"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1024"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1005"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>100</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1006"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1129"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>4</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1128"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>1512</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1130"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1153"/>
</COMPARAM_REF>
</PROTOCOL>
<PROTOCOL EID="ID54">
  <ID>304</ID>
  <SHORT_NAME>SAE_J1939_73_on_SAE_J1939_21</SHORT_NAME>
  <DESCRIPTION>SAE J1939 protocol</DESCRIPTION>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1002"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <COMPARAM IDREF="ID1003"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <MIN_VALUE></MIN_VALUE>
    <COMPARAM IDREF="ID1035"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1005"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1024"/>
  </COMPARAM_REF>
  <COMPARAM_REF>

```

```

    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1026"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <COMPARAM IDREF="ID1015"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1000</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1110"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>2100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1112"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>1500</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1131"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>400</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1133"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>255</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1121"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1143"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1153"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>400</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1160"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>2500</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1161"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>2500</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1162"/>
</COMPARAM_REF>
</PROTOCOL>
<PROTOCOL EID="ID51">
    <ID>301</ID>
    <SHORT_NAME>SAE_J2610_on_SAE_J2610_SCI</SHORT_NAME>
    <DESCRIPTION>SCI Protocol (SAE J2610)</DESCRIPTION>
    <COMPARAM_REF>
        <DEFAULT_VALUE>0</DEFAULT_VALUE>
        <COMPARAM IDREF="ID1002"/>
    </COMPARAM_REF>
</COMPARAM_REF>

```

```

    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1005"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1024"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1153"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1158"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>200</DEFAULT_VALUE>
    <MAX_VALUE></MAX_VALUE>
    <COMPARAM IDREF="ID1159"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1160"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1161"/>
</COMPARAM_REF>
<COMPARAM_REF>
    <DEFAULT_VALUE>200</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1162"/>
</COMPARAM_REF>
</PROTOCOL>
<BUSTYPE EID="ID3">
    <ID>200</ID>
    <SHORT_NAME>ISO_9141_2_UART</SHORT_NAME>
    <DESCRIPTION>K-Line interface as defined in ISO 9141</DESCRIPTION>
    <COMPARAM_REF>
        <DEFAULT_VALUE>10400</DEFAULT_VALUE>
        <MAX_VALUE></MAX_VALUE>
        <COMPARAM IDREF="ID1400"/>
    </COMPARAM_REF>
    <COMPARAM_REF>
        <DEFAULT_VALUE>0</DEFAULT_VALUE>
        <COMPARAM IDREF="ID1403"/>
    </COMPARAM_REF>
    <COMPARAM_REF>
        <DEFAULT_VALUE>0</DEFAULT_VALUE>
        <COMPARAM IDREF="ID1404"/>
    </COMPARAM_REF>
    <COMPARAM_REF>
        <DEFAULT_VALUE>6</DEFAULT_VALUE>
        <COMPARAM IDREF="ID1415"/>
    </COMPARAM_REF>
</BUSTYPE>
<BUSTYPE EID="ID4">
    <ID>201</ID>

```

```

<SHORT_NAME>ISO_11898_2_DWCAN</SHORT_NAME>
<DESCRIPTION>CAN according to ISO 11898</DESCRIPTION>
<COMPARAM_REF>
  <DEFAULT_VALUE>500000</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1400"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>80</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1401"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1405"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1407"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>15</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1412"/>
</COMPARAM_REF>
<COMPARAM_REF>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <COMPARAM IDREF="ID1414"/>
</COMPARAM_REF>
</BUSTYPE>
<BUSTYPE EID="ID5">
  <ID>201</ID>
  <SHORT_NAME>SAE_J2411_SWCAN</SHORT_NAME>
  <DESCRIPTION>CAN according to ISO SAE J2411</DESCRIPTION>
  <COMPARAM_REF>
    <DEFAULT_VALUE>33333</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1400"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>87</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1401"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1405"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1407"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>83333</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1409"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1410"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>2</DEFAULT_VALUE>

```

```

    <COMPARAM IDREF="ID1411"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>15</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1412"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1414"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <COMPARAM IDREF="ID1416"/>
  </COMPARAM_REF>
</BUSTYPE>
<BUSTYPE EID="ID6">
  <ID>203</ID>
  <SHORT_NAME>SCI_J2610_UART</SHORT_NAME>
  <DESCRIPTION>SCI J2610 interface</DESCRIPTION>
  <COMPARAM_REF>
    <DEFAULT_VALUE>7812</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1400"/>
  </COMPARAM_REF>
  <COMPARAM_REF>
    <DEFAULT_VALUE>6</DEFAULT_VALUE>
    <COMPARAM IDREF="ID1415"/>
  </COMPARAM_REF>
</BUSTYPE>
<IO_CTRL EID="ID2199">
  <ID>800</ID>
  <SHORT_NAME>PDU_IOCTL_READ_PROG_VOLTAGE</SHORT_NAME>
  <DESCRIPTION>Read Programming Voltage</DESCRIPTION>
</IO_CTRL>
<IO_CTRL EID="ID2200">
  <ID>801</ID>
  <SHORT_NAME>PDU_IOCTL_SET_PROG_VOLTAGE</SHORT_NAME>
  <DESCRIPTION>Set Programming Voltage</DESCRIPTION>
</IO_CTRL>
<COMPARAM EID="ID1001">
  <ID>1</ID>
  <SHORT_NAME>CP_CanTransmissionTime</SHORT_NAME>
  <DESCRIPTION>If the timeout values are used which have been received by the ECU via session control response (0x50), the Can transmission time has to be added to the timeout values: P2 = received P2 + CanTransmissionTime (contains delay for both transmission directions)</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>200</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1002">
  <ID>2</ID>
  <SHORT_NAME>CP_EnablePerformanceTest</SHORT_NAME>
  <DESCRIPTION>This parameter will place the tester into a performance measurement mode. Parameters such as P1Min, P2Min, Br, Cs will be tested in this mode.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>

```

```

<COMPARAM EID="ID1003">
  <ID>3</ID>
  <SHORT_NAME>CP_J1939Name_Ecu</SHORT_NAME>
  <DESCRIPTION>Name field from J1939 document. This parameter will contain the NAME of the
ECU</DESCRIPTION>
  <DATA_TYPE>PDU_PT_BYTEFIELD</DATA_TYPE>
  <DEFAULT_VALUE>SAE_J1939</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1004">
  <ID>4</ID>
  <SHORT_NAME>CP_J1939PreferredAddress_Ecu</SHORT_NAME>
  <DESCRIPTION>List of preferred addresses for the ECU</DESCRIPTION>
  <DATA_TYPE>PDU_PT_BYTEFIELD</DATA_TYPE>
  <DEFAULT_VALUE>SAE J1939</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER> TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1005">
  <ID>5</ID>
  <SHORT_NAME>CP_Loopback</SHORT_NAME>
  <DESCRIPTION>Echo Transmitted messages in the receive queue. Including periodic messages. Loopback
messages must only be sent after successful transmission of a message. Loopback frames are not
subject to message filtering.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1006">
  <ID>6</ID>
  <SHORT_NAME>CP_P2Max</SHORT_NAME>
  <DESCRIPTION>Timeout in receiving an expected frame after a successful transmit complete. Also
used for multiple ECU responses. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>200</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>APPLICATION</LAYER>
  <RESOLUTION>0.5ms</RESOLUTION>
</COMPARAM>
<COMPARAM EID="ID1007">
  <ID>7</ID>
  <SHORT_NAME>CP_P2Max_Ecu</SHORT_NAME>
  <DESCRIPTION>Performance requirement for the server to start with the response message after the
reception of a request message (indicated via N_USData.ind). This is a performance requirement
parameter.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>200</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1008">
  <ID>8</ID>
  <SHORT_NAME>CP_P2Min</SHORT_NAME>
  <DESCRIPTION>This sets the minimum time between tester request and ECU responses or two ECU
responses. After the request, the interface shall be capable of handling an immediate response
(P2_min=0). For subsequent responses, a byte received after P1_MAX shall be considered as the start
of the subsequent response. This is a performance requirement parameter.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>

```

```

    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1009">
    <ID>9</ID>
    <SHORT_NAME>CP_P2Star</SHORT_NAME>
    <DESCRIPTION>Performance requirement for the client to expect the start of the response message
after the reception of a negative response message (indicated via N_USData.ind) with response code 78
hex (enhanced response timing).</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>12000</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1010">
    <ID>10</ID>
    <SHORT_NAME>CP_P2Star_Ecu</SHORT_NAME>
    <DESCRIPTION>Performance requirement for the server to start with the response message after the
transmission of a negative response message (indicated via N_USData.con) with response code 78 hex
(enhanced response timing). This is a performance requirement parameter.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>10000</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1011">
    <ID>11</ID>
    <SHORT_NAME>CP_P3Func</SHORT_NAME>
    <DESCRIPTION>Minimum time for the client to wait after the successful transmission of a
functionally addressed request message (indicated via N_USData.con) before it can transmit the next
functionally addressed request message in case no response is required or the requested data is only
supported by a subset of the functionally addressed servers.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1012">
    <ID>12</ID>
    <SHORT_NAME>CP_P3Max_Ecu</SHORT_NAME>
    <DESCRIPTION>Time between end of ECU responses and start of new tester request</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>10000</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1013">
    <ID>13</ID>
    <SHORT_NAME>CP_P3Min</SHORT_NAME>
    <DESCRIPTION>Minimum time between end of ECU responses and start of new request. The interface
will accept all responses up to P3_MIN time. The interface will allow transmission of a request any
time after P3_MIN.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>110</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1014">
    <ID>14</ID>

```

```

    <SHORT_NAME>CP_P3Phys</SHORT_NAME>
    <DESCRIPTION>Minimum time for the client to wait after the successful transmission of a physically
addressed request message (indicated via N_USData.con) with no response required before it can
transmit the next physically addressed request message</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>100</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1015">
    <ID>15</ID>
    <SHORT_NAME>CP_J1939PreferredAddress</SHORT_NAME>
    <DESCRIPTION>Claim and protect a SAE J1939 address on the vehicle network. Reference RP1210a.
This will be used by the VCI to claim an address on the J1939 bus</DESCRIPTION>
    <DATA_TYPE>PDU_PT_BYTEFIELD</DATA_TYPE>
    <DEFAULT_VALUE>SAE_J1939</DEFAULT_VALUE>
    <CLASS>INIT</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1016">
    <ID>16</ID>
    <SHORT_NAME>CP_RC21CompletionTimeout</SHORT_NAME>
    <DESCRIPTION>Time period the tester accepts repeated negative responses with response code 0x21
and repeats the same request. </DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>240</DEFAULT_VALUE>
    <CLASS>ERRHDL</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1017">
    <ID>17</ID>
    <SHORT_NAME>CP_RC21Handling</SHORT_NAME>
    <DESCRIPTION>Repetition mode in case of response code 0x7F XX 0x21.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>3</DEFAULT_VALUE>
    <CLASS>ERRHDL</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1018">
    <ID>18</ID>
    <SHORT_NAME>CP_RC21RequestTime</SHORT_NAME>
    <DESCRIPTION>Time between negative response with response code 0x21 and the retransmission of the
same request.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>20</DEFAULT_VALUE>
    <CLASS>ERRHDL</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1019">
    <ID>19</ID>
    <SHORT_NAME>CP_RC23CompletionTimeout</SHORT_NAME>
    <DESCRIPTION>Time period the tester accepts repeated negative responses with response code 0x23
and repeats the same request </DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>ERRHDL</CLASS>
    <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1020">

```

```

<ID>20</ID>
<SHORT_NAME>CP_RC23Handling</SHORT_NAME>
<DESCRIPTION>Repetition mode in case of response code 0x7F XX 0x23. </DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>3</DEFAULT_VALUE>
<CLASS>ERRHDL</CLASS>
<LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1021">
  <ID>21</ID>
  <SHORT_NAME>CP_RC23RequestTime</SHORT_NAME>
  <DESCRIPTION>Time between negative response with response code 0x23 and the retransmission of the
  same request.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>ERRHDL</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1022">
  <ID>22</ID>
  <SHORT_NAME>CP_RC78CompletionTimeout</SHORT_NAME>
  <DESCRIPTION>Time period the tester accepts repeated negative responses with response code 0x78
  and waits for a positive response further on. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>50000</DEFAULT_VALUE>
  <CLASS>ERRHDL</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1023">
  <ID>23</ID>
  <SHORT_NAME>CP_RC78Handling</SHORT_NAME>
  <DESCRIPTION>Handling of 0x7F XX 0x78ResponseTimeout and 0x78Repetitions</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>ERRHDL</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1024">
  <ID>24</ID>
  <SHORT_NAME>CP_RepeatReqCountApp</SHORT_NAME>
  <DESCRIPTION>This parameter contains a counter to enable a re-transmission of the last request
  when either a transmit, receive error, or timeout with no response is detected. This only applies to
  the application layer.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>ERRHDL</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1025">
  <ID>25</ID>
  <SHORT_NAME>CP_StartMsgIndEnable</SHORT_NAME>
  <DESCRIPTION>Start Message Indication Enable. Upon receiving a first frame of a multi-frame
  message (ISO15765) or upon receiving a first byte of a UART message and indication will be set in the
  RX result item. No data bytes will accompany the result item.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>

```

```

<COMPARAM EID="ID1026">
  <ID>26</ID>
  <SHORT_NAME>CP_SuspendQueueOnError</SHORT_NAME>
  <DESCRIPTION>This parameter is to be used as a temporary parameter for services that require a
positive response before any further Com Primitives can be executed.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>ERRHDL</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1027">
  <ID>27</ID>
  <SHORT_NAME>CP_TesterPresentAddrMode</SHORT_NAME>
  <DESCRIPTION>Addressing Mode to be used for Tester Present. Uses the PhysicalReqxxx or FuncReqxxx
information from the address parameter table.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1028">
  <ID>28</ID>
  <SHORT_NAME>CP_TesterPresentHandling</SHORT_NAME>
  <DESCRIPTION>Define tester present message generation settings </DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1031">
  <ID>31</ID>
  <SHORT_NAME>CP_TesterPresentSendType</SHORT_NAME>
  <DESCRIPTION>Define settings for the type of tester present transmits.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1033">
  <ID>33</ID>
  <SHORT_NAME>CP_TesterPresentTime_Ecu</SHORT_NAME>
  <DESCRIPTION>Time for the server to keep a diagnostic session other than the default session
active while not receiving any diagnostic request message</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>10000</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1034">
  <ID>34</ID>
  <SHORT_NAME>CP_TransmitIndEnable</SHORT_NAME>
  <DESCRIPTION>Transmit Indication Enable. On completion of a transmit message by the protocol an
indication will be set in the RX_FLAG result item. No data bytes will accompany the result
item.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>

```

```

<COMPARAM EID="ID1035">
  <ID>35</ID>
  <SHORT_NAME>CP_J1939Name</SHORT_NAME>
  <DESCRIPTION>Name field from J1939 document. This ComParam will contain the NAME of the
Tester.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_BYTEFIELD</DATA_TYPE>
  <DEFAULT_VALUE>SAE_J1939</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER>APPLICATION</LAYER>
</COMPARAM>
<COMPARAM EID="ID1100">
  <ID>100</ID>
  <SHORT_NAME>CP_5BaudAddressFunc</SHORT_NAME>
  <DESCRIPTION>Value of 5Baud Address in case of functional addressed communication. The correct
baud rate address type (functional/physical) is selected during execution of a Start Communication
Com Primitive based on the setting of the CP_RequestAddrMode parameter</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>51</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1101">
  <ID>101</ID>
  <SHORT_NAME>CP_5BaudAddressPhys</SHORT_NAME>
  <DESCRIPTION>Value of 5Baud Address in case of physical addressed communication. The correct baud
rate address type (functional/physical) is selected during execution of a Start Communication Com
Primitive based on the setting of the CP_RequestAddrMode parameter</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>01</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1102">
  <ID>102</ID>
  <SHORT_NAME>CP_5BaudMode</SHORT_NAME>
  <DESCRIPTION>Type of 5 Baud initialization. This parameter allows either ISO9141 initialization
sequence, ISO9141-2/ISO14230 initialization sequence, or hybrid versions which include only one of
the extra bytes defined for ISO9141-2 and ISO14230 </DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1103">
  <ID>103</ID>
  <SHORT_NAME>CP_Ar</SHORT_NAME>
  <DESCRIPTION>Time for transmission of the CAN frame (any N_PDU) on the receiver side</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>50</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1104">
  <ID>104</ID>
  <SHORT_NAME>CP_Ar_Ecu</SHORT_NAME>
  <DESCRIPTION>Time for transmission of the CAN frame (any N_PDU) on the receiver side</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>50</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>

```

```

    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1105">
    <ID>105</ID>
    <SHORT_NAME>CP_As</SHORT_NAME>
    <DESCRIPTION>Time for transmission of the CAN frame (any N_PDU) on the sender side</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>50</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1106">
    <ID>106</ID>
    <SHORT_NAME>CP_As_Ecu</SHORT_NAME>
    <DESCRIPTION>Time for transmission of the CAN frame (any N_PDU) on the sender side</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>50</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1107">
    <ID>107</ID>
    <SHORT_NAME>CP_BlockSize</SHORT_NAME>
    <DESCRIPTION>This sets the block size the interface should report to the vehicle for receiving
segmented transfers in a Transmit Flow Control Message.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1108">
    <ID>108</ID>
    <SHORT_NAME>CP_BlockSize_Ecu</SHORT_NAME>
    <DESCRIPTION>This sets the block size the ECU should report to the tester for receiving segmented
transfers in a Transmit Flow Control Message.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1109">
    <ID>109</ID>
    <SHORT_NAME>CP_BlockSizeOverride</SHORT_NAME>
    <DESCRIPTION>This sets the block size the interface should use to send segmented messages to the
vehicle. The flow control value reported by the vehicle should be ignored. </DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>65535</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1110">
    <ID>110</ID>
    <SHORT_NAME>CP_Br</SHORT_NAME>
    <DESCRIPTION>Time until transmission of the next FlowControl. This is equivalent to Th in J1939-
21. For ISO 15765-2 and 15765-4, this value is a performance requirement parameter and should not be
used as a timeout value by the tester.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>20</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>

```

```

    <LAYER>TRANSPORT</LAYER>
  </COMPARAM>
  <COMPARAM EID="ID1111">
    <ID>111</ID>
    <SHORT_NAME>CP_Br_Ecu</SHORT_NAME>
    <DESCRIPTION>Time until transmission of the next FlowControl. This is a performance requirement parameter.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>20</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
  </COMPARAM>
  <COMPARAM EID="ID1112">
    <ID>112</ID>
    <SHORT_NAME>CP_Bs</SHORT_NAME>
    <DESCRIPTION>Timeout until reception of the next FlowControl. This is equivalent to T4 in J1939-21.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>150</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
  </COMPARAM>
  <COMPARAM EID="ID1113">
    <ID>113</ID>
    <SHORT_NAME>CP_Bs_Ecu</SHORT_NAME>
    <DESCRIPTION>Timeout until reception of the next FlowControl</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>150</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
  </COMPARAM>
  <COMPARAM EID="ID1114">
    <ID>114</ID>
    <SHORT_NAME>CP_CanDataSizeOffset</SHORT_NAME>
    <DESCRIPTION>Offset subtracted from the total number of expected bytes received/transmitted in a first frame message.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
  </COMPARAM>
  <COMPARAM EID="ID1115">
    <ID>115</ID>
    <SHORT_NAME>CP_CanFillerByte</SHORT_NAME>
    <DESCRIPTION>Padding data byte to be used to pad all USDT type transmits frames (SF, FC, and last CF).</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>00</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
  </COMPARAM>
  <COMPARAM EID="ID1116">
    <ID>116</ID>
    <SHORT_NAME>CP_CanFillerByteHandling</SHORT_NAME>
    <DESCRIPTION>Enable Padding forcing the DLC of a CAN frame to always be 8.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>1</DEFAULT_VALUE>
    <CLASS>COM</CLASS>

```

```

<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1117">
  <ID>117</ID>
  <SHORT_NAME>CP_CanFirstFrameValue</SHORT_NAME>
  <DESCRIPTION>First Frame number to be transmitted/received on a multi-segment transfer. Used to
  override the normal First Frame value of 1</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>1</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1118">
  <ID>118</ID>
  <SHORT_NAME>CP_CanFuncReqExtAddr</SHORT_NAME>
  <DESCRIPTION>Address extension for enhanced diagnostics. The first byte of the requested CAN
  frame data contains the N_AE/N_TA byte followed by the correct number of PCI bytes. This parameters
  is used for all transmitted CAN Frames that have the "Can Address Extension' Aerial bit set in the
  CanIdFormat</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1119">
  <ID>119</ID>
  <SHORT_NAME>CP_CanFuncReqFormat</SHORT_NAME>
  <DESCRIPTION>CAN Format used for a functional address transmit</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>05</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1120">
  <ID>120</ID>
  <SHORT_NAME>CP_CanFuncReqId</SHORT_NAME>
  <DESCRIPTION>CAN ID used for a functional address transmit</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>2015</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1121">
  <ID>121</ID>
  <SHORT_NAME>CP_CanMaxNumWaitFrames</SHORT_NAME>
  <DESCRIPTION>The maximum number of WAIT flow control frames allowed during a multi-segment
  transfer. For J1939, this is the maximum number of allowed CTS frames.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>255</DEFAULT_VALUE>
  <CLASS>COM</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1122">
  <ID>122</ID>
  <SHORT_NAME>CP_CanPhysReqExtAddr</SHORT_NAME>
  <DESCRIPTION>Address extension for enhanced diagnostics. The first byte of the requested CAN
  frame data contains the N_AE/N_TA byte followed by the correct number of PCI bytes. This parameters
  is used for all transmitted CAN Frames that have the "Can Address Extension' bit set in the
  CanIdFormat</DESCRIPTION>

```

```

<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>00</DEFAULT_VALUE>
<CLASS>UNIQUE_ID</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1123">
  <ID>123</ID>
  <SHORT_NAME>CP_CanPhysReqFormat</SHORT_NAME>
  <DESCRIPTION>CAN Format used for a physical address transmit</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>05</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1124">
  <ID>124</ID>
  <SHORT_NAME>CP_CanPhysReqId</SHORT_NAME>
  <DESCRIPTION>CAN ID used for a physical address transmit</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>2016</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1125">
  <ID>125</ID>
  <SHORT_NAME>CP_CanRespUSDTExtAddr</SHORT_NAME>
  <DESCRIPTION>Extended Address used for a USDT response from an ECU if the CAN Format indicates
address extension</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1126">
  <ID>126</ID>
  <SHORT_NAME>CP_CanRespUSDTFormat</SHORT_NAME>
  <DESCRIPTION>CAN Format for the USDT CAN ID received from an ECU (Segment type Bit must =
1)</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>05</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1127">
  <ID>127</ID>
  <SHORT_NAME>CP_CanRespUSDTId</SHORT_NAME>
  <DESCRIPTION>Received USDT CAN ID from an ECU</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>2024</DEFAULT_VALUE>
  <CLASS>UNIQUE_ID</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1128">
  <ID>128</ID>
  <SHORT_NAME>CP_CanRespUUDTFormat</SHORT_NAME>
  <DESCRIPTION>Received CAN Format for CAN ID without segmentation (Segment Type Bit must =
0)</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>

```

```

    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>UNIQUE_ID</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1129">
    <ID>129</ID>
    <SHORT_NAME>CP_CanRespUUDTextAddr</SHORT_NAME>
    <DESCRIPTION>Extended Address used for UUDT response if the CAN Format indicates address
extension</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>UNIQUE_ID</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1130">
    <ID>130</ID>
    <SHORT_NAME>CP_CanRespUUDTId</SHORT_NAME>
    <DESCRIPTION>Received UUDT CAN ID from an ECU</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>1512</DEFAULT_VALUE>
    <CLASS>UNIQUE_ID</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1131">
    <ID>131</ID>
    <SHORT_NAME>CP_Cr</SHORT_NAME>
    <DESCRIPTION>Timeout for reception of the next ConsecutiveFrame. For J1939-21, this is
equivalent to T1.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>300</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1132">
    <ID>132</ID>
    <SHORT_NAME>CP_Cr_Ecu</SHORT_NAME>
    <DESCRIPTION>Timeout for reception of the next ConsecutiveFrame</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>300</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1133">
    <ID>133</ID>
    <SHORT_NAME>CP_Cs</SHORT_NAME>
    <DESCRIPTION>Time until transmission of the next Consecutive Frame (This is used if FC is not
enabled or if the STmin value in the FC=0 and STminOverride=0xFFFF). See ISO 15765-2. For ISO
15765-2 and 15765-4, this is a performance requirement parameter and should not be used as a timeout
value by the tester. For J1939, this is equivalent to the maximum time between sending packets in a
multi-packet broadcast and multi-packet destination specific message. From text in J1939-21 section
5.12.3. </DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>20</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1134">
    <ID>134</ID>
    <SHORT_NAME>CP_Cs_Ecu</SHORT_NAME>

```

```

<DESCRIPTION>Time until transmission of the next Consecutive Frame (This is used if FC is not
enabled or if the STmin value in the FC=0 and STminOverride=0xFFFF). See ISO 15765-2. This is a
performance requirement parameter.</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>20</DEFAULT_VALUE>
<CLASS>TIMING</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1135">
<ID>135</ID>
<SHORT_NAME>CP_EcuRespSourceAddress</SHORT_NAME>
<DESCRIPTION>ECU Source Address response of a non-CAN message.</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>16</DEFAULT_VALUE>
<CLASS>UNIQUE_ID</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1136">
<ID>136</ID>
<SHORT_NAME>CP_FuncReqFormatPriorityType</SHORT_NAME>
<DESCRIPTION>First Header Byte of a non-CAN message for a functional address
transmit</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>104</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1137">
<ID>137</ID>
<SHORT_NAME>CP_FuncRespFormatPriorityType</SHORT_NAME>
<DESCRIPTION>First Header Byte of a non-CAN message received from the ECU for functional
addressing</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>72</DEFAULT_VALUE>
<CLASS>UNIQUE_ID</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1138">
<ID>138</ID>
<SHORT_NAME>CP_FuncReqTargetAddr</SHORT_NAME>
<DESCRIPTION>Second Header Byte of a non-CAN message for a functional address
transmit</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>106</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1139">
<ID>139</ID>
<SHORT_NAME>CP_FuncRespTargetAddr</SHORT_NAME>
<DESCRIPTION>Second Header Byte of a non-CAN message received from the ECU for functional
addressing. This information is also used to fill out the functional lookup table for
J1850_PWM.</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>107</DEFAULT_VALUE>
<CLASS>UNIQUE_ID</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1140">

```

```

<ID>140</ID>
<SHORT_NAME>CP_HeaderFormatJ1850</SHORT_NAME>
<DESCRIPTION>Header Byte configuration to be used for J1850 communication. This setting is used to properly construct the message header bytes to complete the PDU. This parameter is not used if the protocol parameter RawMode is set. Header bytes are constructed following the rules of the protocol specification </DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>3</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1141">
<ID>141</ID>
<SHORT_NAME>CP_HeaderFormatKW</SHORT_NAME>
<DESCRIPTION>Header Byte configuration for K-Line protocol (Keyword). This setting is used to properly construct the message header bytes to complete the PDU. This parameter is not used if the protocol parameter RawMode is set. Header bytes are constructed following the rules of the protocol specification. This parameter overrides any keybyte values received from the ECU during initialization, which could be used for automatic header byte construction.</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>4</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1142">
<ID>142</ID>
<SHORT_NAME>CP_InitializationSettings</SHORT_NAME>
<DESCRIPTION>Set Initialization method. </DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>1</DEFAULT_VALUE>
<CLASS>INIT</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1143">
<ID>143</ID>
<SHORT_NAME>CP_MessagePriority</SHORT_NAME>
<DESCRIPTION>Message Priority. J1939 protocol uses the 3 least significant bits that become part of the CAN ID. J1708 uses 8 bits to define the first byte of the transmit message</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>0</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1144">
<ID>144</ID>
<SHORT_NAME>CP_MidReqId</SHORT_NAME>
<DESCRIPTION>Request Message Identifier used in building a transmit message to an ECU for a J1708 protocol only.</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>0</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1145">
<ID>145</ID>
<SHORT_NAME>CP_MidRespId</SHORT_NAME>
<DESCRIPTION>Response Message Identifier received from an ECU for a J1708 protocol only.</DESCRIPTION>
<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>0</DEFAULT_VALUE>

```

```

    <CLASS>UNIQUE_ID</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1146">
    <ID>146</ID>
    <SHORT_NAME>CP_P1Max</SHORT_NAME>
    <DESCRIPTION>Maximum inter-byte time for ECU Responses. Interface must be capable of handling a P1_MIN time of 0 ms. After the request, the interface shall be capable of handling an immediate response (P2_MIN=0). For subsequent responses, a byte received after P1_MAX shall be considered as the start of the subsequent response.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1147">
    <ID>147</ID>
    <SHORT_NAME>CP_P1Min</SHORT_NAME>
    <DESCRIPTION>This sets the minimum inter-byte time for the ECU responses. Application shall not get or set this value. Interface must be capable of handling P1_MIN=0. This is a performance requirement parameter.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1148">
    <ID>148</ID>
    <SHORT_NAME>CP_P4Max</SHORT_NAME>
    <DESCRIPTION>Maximum inter-byte time for a tester request.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>40</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1149">
    <ID>149</ID>
    <SHORT_NAME>CP_P4Min</SHORT_NAME>
    <DESCRIPTION>Minimum inter-byte time for tester transmits.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>10</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1150">
    <ID>145</ID>
    <SHORT_NAME>CP_PhysReqFormatPriorityType</SHORT_NAME>
    <DESCRIPTION>First Header Byte of a non-CAN message for physical address transmit</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>108</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1151">
    <ID>151</ID>
    <SHORT_NAME>CP_PhysRespFormatPriorityType</SHORT_NAME>
    <DESCRIPTION>First Header Byte of a non-CAN message received from the ECU for physical addressing</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>

```

```

    <DEFAULT_VALUE>104</DEFAULT_VALUE>
    <CLASS>UNIQUE_ID</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1152">
    <ID>152</ID>
    <SHORT_NAME>CP_PhysReqTargetAddr</SHORT_NAME>
    <DESCRIPTION>Physical Target Addressing Information used for correct Message Header Construction
</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>16</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1153">
    <ID>153</ID>
    <SHORT_NAME>CP_RepeatReqCountTrans</SHORT_NAME>
    <DESCRIPTION>This parameter contains a counter to enable a re-transmission of the last request
when either a transmit, a receive error, or transport layer timeout is detected. This applies to the
transport layer only.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>ERRHDL</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1154">
    <ID>154</ID>
    <SHORT_NAME>CP_RequestAddrMode</SHORT_NAME>
    <DESCRIPTION>Addressing Mode to be used for the Com Primitive</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>2</DEFAULT_VALUE>
    <CLASS>COM</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1155">
    <ID>155</ID>
    <SHORT_NAME>CP_StMin</SHORT_NAME>
    <DESCRIPTION>This sets the separation time the interface should report to the vehicle for
receiving segmented transfers in a Transmit Flow Control Message.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1156">
    <ID>156</ID>
    <SHORT_NAME>CP_StMin_Ecu</SHORT_NAME>
    <DESCRIPTION>The minimum time the sender shall wait between the transmissions of two
ConsecutiveFrame N_PDUs</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>TIMING</CLASS>
    <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1157">
    <ID>157</ID>
    <SHORT_NAME>CP_StMinOverride</SHORT_NAME>
    <DESCRIPTION>This sets the separation time the interface should use to transmit segmented messages
to the vehicle. The flow control value reported by the vehicle should be ignored</DESCRIPTION>

```

```

<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>65535</DEFAULT_VALUE>
<CLASS>TIMING</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1158">
  <ID>158</ID>
  <SHORT_NAME>CP_T1Max</SHORT_NAME>
  <DESCRIPTION>This sets the maximum inter-frame response delay.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>40</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1159">
  <ID>159</ID>
  <SHORT_NAME>CP_T2Max</SHORT_NAME>
  <DESCRIPTION>This sets the maximum inter-frame request delay.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>200</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1160">
  <ID>160</ID>
  <SHORT_NAME>CP_T3Max</SHORT_NAME>
  <DESCRIPTION>This sets the maximum response delay from the ECU after processing a valid request
  message from the interface. For J1939-21, this is equivalent to Tr.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>100</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1161">
  <ID>161</ID>
  <SHORT_NAME>CP_T4Max</SHORT_NAME>
  <DESCRIPTION>This sets the maximum inter-message response delay. For J1939, this is equivalent to
  T3, the maximum time allowed for the Originator to receive a CTS or an ACK after sending a packet.
  </DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>40</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1162">
  <ID>162</ID>
  <SHORT_NAME>CP_T5Max</SHORT_NAME>
  <DESCRIPTION>This sets the maximum inter-message request delay. For J1939, this is equivalent to
  T2, the maximum time allowed for the Originator to send a packet after receiving a CTS from the
  Responder. </DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>200</DEFAULT_VALUE>
  <CLASS>TIMING</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1163">
  <ID>163</ID>
  <SHORT_NAME>CP_TesterSourceAddress</SHORT_NAME>
  <DESCRIPTION>Source address of transmitted message for non-CAN messages</DESCRIPTION>

```

```

<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>241</DEFAULT_VALUE>
<CLASS>COM</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1164">
  <ID>164</ID>
  <SHORT_NAME>CP_Tidle</SHORT_NAME>
  <DESCRIPTION>Minimum bus idle time before tester starts the address byte sequence or the fast init
sequence. (TIDLE replaces W0 and W5).</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>600</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1165">
  <ID>165</ID>
  <SHORT_NAME>CP_TInil</SHORT_NAME>
  <DESCRIPTION>Sets the duration for the low pulse in a fast initialization sequence.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>50</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1166">
  <ID>166</ID>
  <SHORT_NAME>CP_TWup</SHORT_NAME>
  <DESCRIPTION>Sets total duration of the wakeup pulse (TWUP-TINIL)=high pulse before start
communication message.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>100</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1167">
  <ID>167</ID>
  <SHORT_NAME>CP_W1Max</SHORT_NAME>
  <DESCRIPTION>Maximum time from the end of address byte to start of the synchronization pattern
from the ECU.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>600</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1168">
  <ID>168</ID>
  <SHORT_NAME>CP_W1Min</SHORT_NAME>
  <DESCRIPTION>Minimum time from the end of address byte to start of the synchronization pattern
from the ECU.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>120</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1169">
  <ID>169</ID>
  <SHORT_NAME>CP_W2Max</SHORT_NAME>
  <DESCRIPTION>Maximum time from the end of the synchronization pattern to the start of key byte
1.</DESCRIPTION>

```

```

<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>40</DEFAULT_VALUE>
<CLASS>INIT</CLASS>
<LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1170">
  <ID>170</ID>
  <SHORT_NAME>CP_W2Min</SHORT_NAME>
  <DESCRIPTION>Minimum time from the end of the synchronization pattern to the start of key byte
1.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>10</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1171">
  <ID>171</ID>
  <SHORT_NAME>CP_W3Max</SHORT_NAME>
  <DESCRIPTION>Maximum time between key byte 1 and key byte 2.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>40</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1172">
  <ID>172</ID>
  <SHORT_NAME>CP_W3Min</SHORT_NAME>
  <DESCRIPTION>Minimum time between key byte 1 and key byte 2.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1173">
  <ID>173</ID>
  <SHORT_NAME>CP_W4Max</SHORT_NAME>
  <DESCRIPTION>Maximum time between receiving key byte 2 from the vehicle and the inversion being
returned by the interface.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>100</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1174">
  <ID>174</ID>
  <SHORT_NAME>CP_W4Min</SHORT_NAME>
  <DESCRIPTION>Minimum time between receiving key byte 2 from the vehicle and the inversion being
returned by the interface.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>50</DEFAULT_VALUE>
  <CLASS>INIT</CLASS>
  <LAYER>TRANSPORT</LAYER>
</COMPARAM>
<COMPARAM EID="ID1400">
  <ID>400</ID>
  <SHORT_NAME>CP_Baudrate</SHORT_NAME>
  <DESCRIPTION>Represents the desired baud rate. If the desired baud rate cannot be achieved within
the tolerance of the protocol, the interface will remain at the previous baud rate.</DESCRIPTION>

```

```

<DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
<DEFAULT_VALUE>500000</DEFAULT_VALUE>
<CLASS>BUSTYPE</CLASS>
<LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1401">
  <ID>401</ID>
  <SHORT_NAME>CP_BitSamplePoint</SHORT_NAME>
  <DESCRIPTION>This sets the desired bit sample point as a percentage of the bit time.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>80</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1402">
  <ID>402</ID>
  <SHORT_NAME>CP_BitSamplePoint_Ecu</SHORT_NAME>
  <DESCRIPTION>This sets the desired bit sample point as a percentage of the bit time.</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE> 80</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1403">
  <ID>403</ID>
  <SHORT_NAME>CP_K_L_LineInit</SHORT_NAME>
  <DESCRIPTION>K & L line usage for ISO9141 and ISO14230 initialization address</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1404">
  <ID>404</ID>
  <SHORT_NAME>CP_K_LinePullup</SHORT_NAME>
  <DESCRIPTION>Control the K-Line voltage to either 12V or 24V</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1405">
  <ID>405</ID>
  <SHORT_NAME>CP_ListenOnly</SHORT_NAME>
  <DESCRIPTION>Enable a Listen Only mode on the Com Logical Link. This will cause the link to no longer acknowledge received frames on the CAN Network</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>
  <CLASS>BUSTYPE</CLASS>
  <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1406">
  <ID>406</ID>
  <SHORT_NAME>CP_NetworkLine</SHORT_NAME>
  <DESCRIPTION>This sets the network line(s) that are active during communication (for cases where the physical layer allows this)</DESCRIPTION>
  <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
  <DEFAULT_VALUE>0</DEFAULT_VALUE>

```

```

    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1407">
    <ID>407</ID>
    <SHORT_NAME>CP_SamplesPerBit</SHORT_NAME>
    <DESCRIPTION>Number of samples per bit</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1408">
    <ID>408</ID>
    <SHORT_NAME>CP_SamplesPerBit_Ecu</SHORT_NAME>
    <DESCRIPTION>Number of samples per bit</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1412">
    <ID>412</ID>
    <SHORT_NAME>CP_SyncJumpWidth</SHORT_NAME>
    <DESCRIPTION>This sets the desired synchronization jump width as a percentage of the bit
time.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>15</DEFAULT_VALUE>
    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1413">
    <ID>413</ID>
    <SHORT_NAME>CP_SyncJumpWidth_Ecu</SHORT_NAME>
    <DESCRIPTION>This sets the desired synchronization jump width as a percentage of the bit
time.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>15</DEFAULT_VALUE>
    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1414">
    <ID>414</ID>
    <SHORT_NAME>CP_TerminationType</SHORT_NAME>
    <DESCRIPTION>CAN termination settings.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>0</DEFAULT_VALUE>
    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>
</COMPARAM>
<COMPARAM EID="ID1415">
    <ID>415</ID>
    <SHORT_NAME>CP_UartConfig</SHORT_NAME>
    <DESCRIPTION>Configure the parity, data bit size and stop bits of a Uart protocol.</DESCRIPTION>
    <DATA_TYPE>PDU_PT_UNUM32</DATA_TYPE>
    <DEFAULT_VALUE>06</DEFAULT_VALUE>
    <CLASS>BUSTYPE</CLASS>
    <LAYER>PHYSICAL</LAYER>

```

```

</COMPARAM>
<ERROR_CODE>
  <ID>3000</ID>
  <SHORT_NAME>AN_ERR_CODE</SHORT_NAME>
  <DESCRIPTION>This is an error code description</DESCRIPTION>
</ERROR_CODE>
</MVCI_MODULE_DESCRIPTION>

```

#### F.5.4 Example cable description file

Since the external connector on the MVCI protocol module may differ from the DLC on the vehicle or ECU setup, there is a need to describe how the cable maps the pins on the MVCI protocol module (PIN\_ON\_MODULE) onto the pins on the DLC (PIN\_ON\_DLC). The cable description file (see EXAMPLE Cable Description File) shows the mapping for two example cables that this MVCI protocol module is supposed to support. Also, if of interest for the application, the cable description file optionally defines which pins of the MVCI protocol module are used for cable identification and what resistor values are expected for a specific cable (see also ISO 22900-1 about cable coding). However, the example only shows the first of both cables.

EXAMPLE Cable description file (CDF file).

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- D-PDU-API cable description file -->
<MVCI_CABLE_DESCRIPTION xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Data\Dev\eclipse\workspace\PduApi\pdu.xsd" FILE_VERSION="0.0.1"
MVCI_PART2_STANDARD_VERSION="2.2.1">
  <DESCRIPTION>This is an example for a cable description file (CDF)</DESCRIPTION>
  <SUPPLIER_NAME>Vehicle Doctor Ltd.</SUPPLIER_NAME>
  <DLCTYPE EID="ID1">
    <ID>1</ID>
    <SHORT_NAME>ISO_15031_3</SHORT_NAME>
    <DESCRIPTION>ISO 15031-3 OBD Connector</DESCRIPTION>
  </DLCTYPE>
  <CABLE>
    <ID>100</ID>
    <SHORT_NAME>SomeCableName</SHORT_NAME>
    <DESCRIPTION>Standard cable 5m</DESCRIPTION>
    <CABLE_IDENTIFICATION>
      <CABLE_ID>1002</CABLE_ID>
      <CABLE_ID_PIN>
        <PIN_ON_MODULE>17</PIN_ON_MODULE>
        <RESISTANCE_TO_GROUND>250</RESISTANCE_TO_GROUND>
      </CABLE_ID_PIN>
      <CABLE_ID_PIN>
        <PIN_ON_MODULE>18</PIN_ON_MODULE>
        <RESISTANCE_TO_GROUND>49999</RESISTANCE_TO_GROUND>
      </CABLE_ID_PIN>
    </CABLE_IDENTIFICATION>
    <MAPPING>
      <PIN_ON_DLC>13</PIN_ON_DLC>
      <PIN_ON_MODULE>5</PIN_ON_MODULE>
    </MAPPING>
    <MAPPING>
      <PIN_ON_DLC>16</PIN_ON_DLC>
      <PIN_ON_MODULE>6</PIN_ON_MODULE>
    </MAPPING>
    <MAPPING>
      <PIN_ON_DLC>15</PIN_ON_DLC>

```

```

    <PIN_ON_MODULE>8</PIN_ON_MODULE>
  </MAPPING>
  <DLCTYPE IDREF="ID1"/>
</CABLE>
<CABLE>
  <ID>101</ID>
  <SHORT_NAME>SomeOtherCableName</SHORT_NAME>
  <DESCRIPTION>Standard cable 5m</DESCRIPTION>
  <CABLE_IDENTIFICATION>
    <CABLE_ID>1002</CABLE_ID>
    <CABLE_ID_PIN>
      <PIN_ON_MODULE>17</PIN_ON_MODULE>
      <RESISTANCE_TO_GROUND>250</RESISTANCE_TO_GROUND>
    </CABLE_ID_PIN>
    <CABLE_ID_PIN>
      <PIN_ON_MODULE>18</PIN_ON_MODULE>
      <RESISTANCE_TO_GROUND>4294967295</RESISTANCE_TO_GROUND>
    </CABLE_ID_PIN>
  </CABLE_IDENTIFICATION>
  <MAPPING>
    <PIN_ON_DLC>13</PIN_ON_DLC>
    <PIN_ON_MODULE>5</PIN_ON_MODULE>
  </MAPPING>
  <MAPPING>
    <PIN_ON_DLC>16</PIN_ON_DLC>
    <PIN_ON_MODULE>6</PIN_ON_MODULE>
  </MAPPING>
  <MAPPING>
    <PIN_ON_DLC>15</PIN_ON_DLC>
    <PIN_ON_MODULE>9</PIN_ON_MODULE>
  </MAPPING>
  <DLCTYPE IDREF="ID1"/>
</CABLE>
</MVCICABLE_DESCRIPTION>

```

STANDARDSISO.COM : Click to view the full PDF of ISO 22900-2:2017

## Annex G (informative)

### Resource handling scenarios

#### G.1 Resource handling at the API level

##### G.1.1 Obtaining resource and object ids

###### G.1.1.1 General

A client application can use an XML parser to parse the CDF and MDF files to obtain object ids and resource ids. It is also possible to obtain the object ids by using the D-PDU API function `PDUGetObjectIds` using the standard short names of the object. For a client application to retrieve a resource id using `PDUGetObjectIds`, the client application would need to know the vendor-specific short name of the resource.

Once a client application obtains the list of ids supported by the specific D-PDU API implementation, all D-PDU API functions can be used. Without the list of ids, many D-PDU API functions cannot be used.

EXAMPLE D-PDU API functions requiring object and or resource ids:

- `PDUCreateComLogicalLink`;
- `PDUSetComParam`;
- `PDUGetResourceStatus`;
- `PDUGetConflictingResources`;
- `PDUSetUniqueRespIdTable`.

###### G.1.1.2 Using an XML parser

Figure G.1 illustrates via a sequence diagram how a client application can use an XML parser for the MDF and CDF files to retrieve resource and object ids. D-PDU API functions still need to be called to retrieve the status of a resource and any conflicts on a resource.

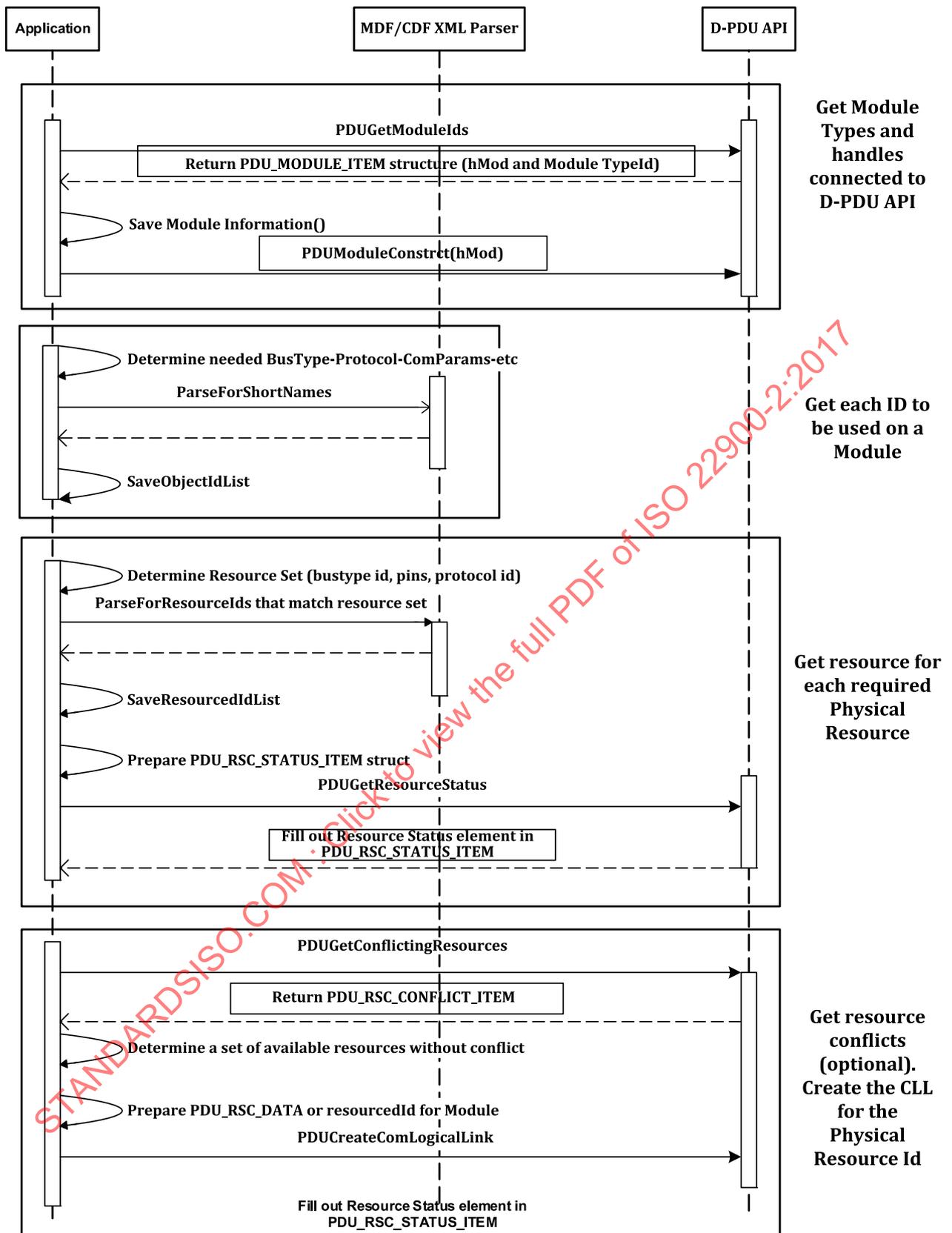


Figure G.1 — Sequence for retrieving ids using an XML parser

G.1.1.3 Using D-PDU API functions for resource information

Figure G.2 illustrates how the D-PDU API functions are used to retrieve resource and object ids without parsing the XML MDF/CDF files. The functions (represented by circles) are expected to be called in the order indicated by the numbers in the circle.

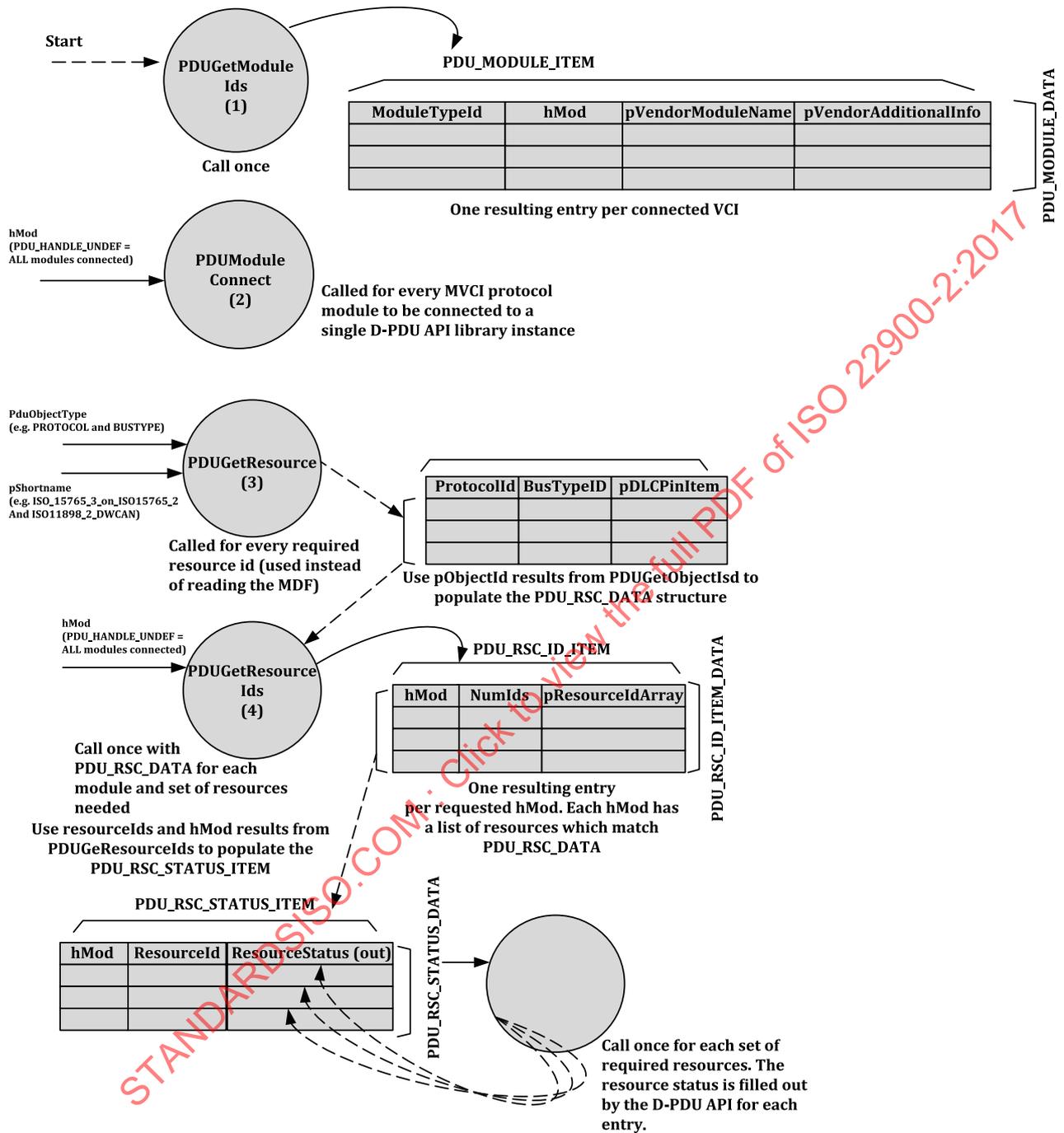


Figure G.2 — Using D-PDU API functions to retrieve resource information and status

G.1.1.4 Retrieving conflicting resources

Figure G.3 illustrates how the D-PDU API supports the client application in determining shared conflicts on a resource.

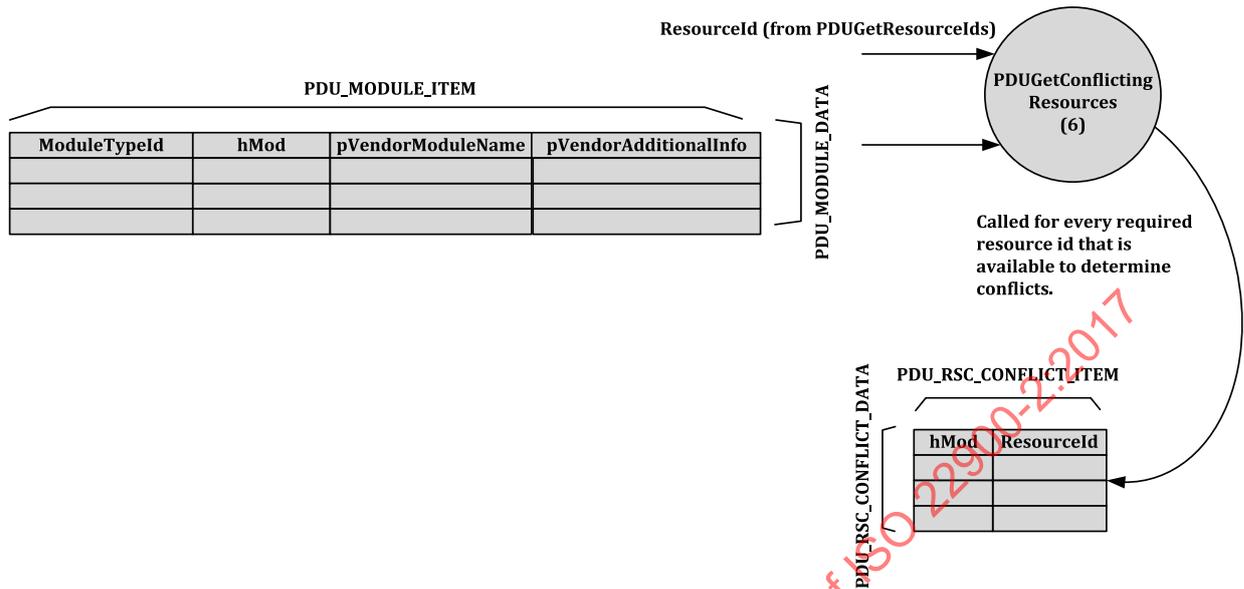


Figure G.3 — Conflicting resources

G.1.2 Example MVCI protocol module resource selection

Figure G.4 shows an example MVCI protocol module and cable for the purposes of demonstrating the logic employed by an application selecting resources.

In the example, the MVCI protocol module includes two general-purpose CAN controllers. The first may be used for either Single Wire or Dual Wire High Speed CAN. The second may be used for either Fault Tolerant or Dual Wire High Speed CAN. The resources functions allow an application to ensure that predictable results are always obtained each time it executes. For example, it is possible that an application could first request a Dual Wire High Speed CAN bus type and be allocated the first CAN controller, and then request a Single Wire CAN bus type. The second request could not be fulfilled due to pre-selection of the only CAN controller that could support Single Wire CAN. If the bus types had been requested in the reverse order, the required resources could have been satisfied.

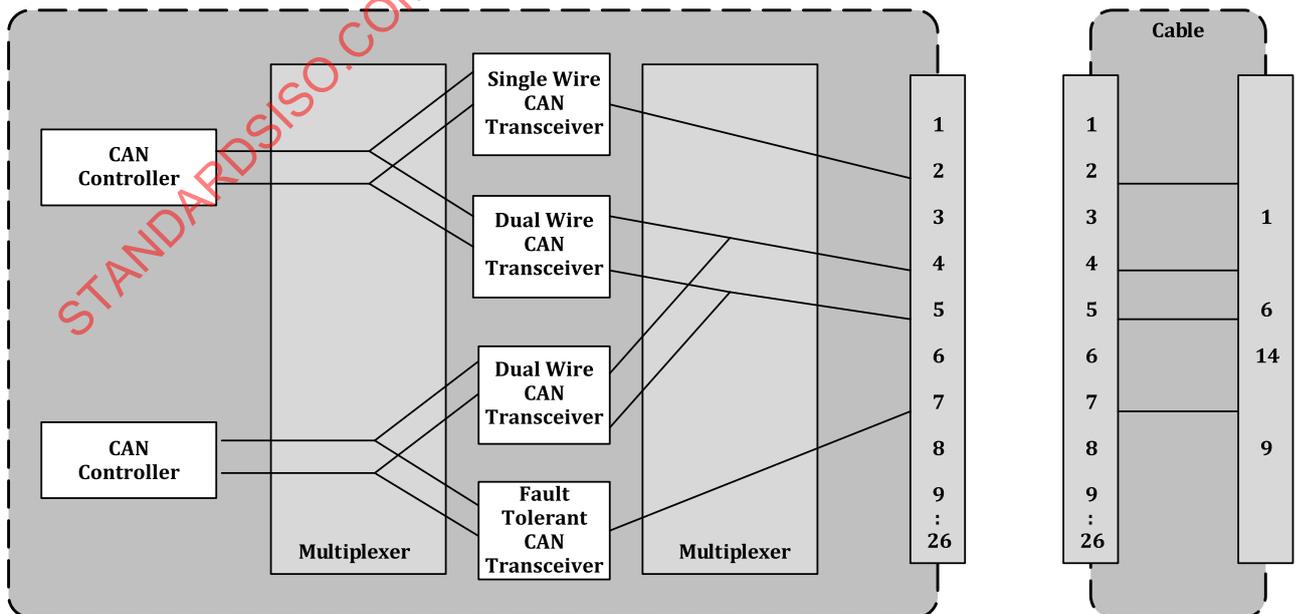


Figure G.4 — Example MVCI protocol module and cable

Figure G.5 shows the four possible combinations of resources supported by this combination of MSCI protocol module and cable. These combinations are known as routes and are represented in the D-PDU API by ResourceIds. The application example requires two routes, one single wire and one dual wire. It uses the GetModuleIDs, GetObjectID and GetResourceIDs to obtain ResourceIds (each representing a route) that support the requirements of each connection. For this MSCI protocol module, there is one route that supports Single Wire CAN (Route 1) and two that support Dual Wire CAN (Route 2 and Route 3). After confirming that all three routes are available using GetResourceStatus, the application has to make a choice between the two possible routes supporting High Speed CAN. It makes this decision by checking for conflicts between the three routes.

GetConflictingResources is called once for each of the three routes.

For Route 1: Route 2 is indicated as conflicting (due to the common CAN Controller).

For Route 2: Routes 1 and Route 3 are indicated as conflicting (Route 1 due to the common CAN controller, Route 3 due to common pins).

For Route 3: Route 2 and Route 4 are conflicting (Route 2 due to common pins and Route 4 due to the common CAN controller).

Since there is only one route (Route 1) that supports Single Wire CAN, the application checks for the Dual Wire Route that does not conflict with Route 1. Hence, Route 2 is discounted due to the indicated conflict and Route 3 is selected.

The application proceeds to call PDUCreateComLogicalLink once using the ResourceId provided for Route 1 and once using the ResourceId provided for Route 3.

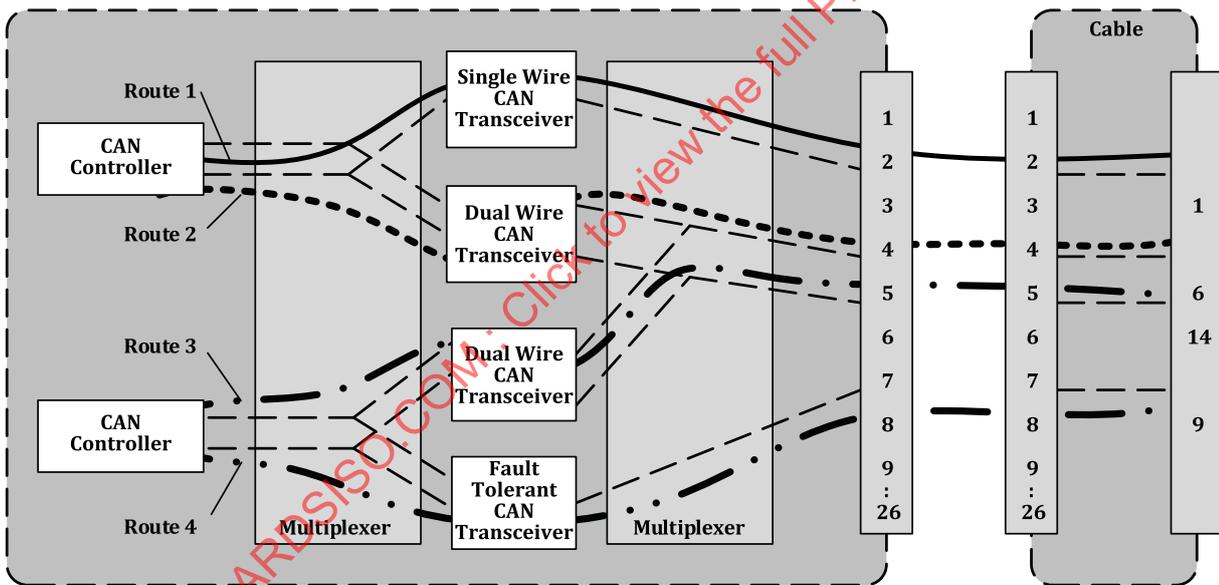


Figure G.5 — Available routes and selection logic

## Annex H (informative)

### D-PDU API partitioning

#### H.1 Functional partitioning of a D-PDU API

##### H.1.1 ODX data base

Provides data describing the vehicle under test, including connector information, protocol information, vehicle network topology, ECU information and vehicle data service information.

##### H.1.2 MSCI D-Server

###### H.1.2.1 Com Primitive Creator/Handler

Requests by an MSCI D-Server Job or application for data retrieval from an ECU. No information about the protocol or ECU is necessary for the requestor. All information about how to generate a D-PDU, check for valid results, extract the data and finally convert the data into correct units is done by the ComPrimitive Creator/Handler.

###### H.1.2.2 Rx Logical Data Request

API for requesting logical data from an ECU.

###### H.1.2.3 D-PDU Builder

Builds a PDU message to be requested from the ECU. The information is data only (header bytes and formatting is accomplished by the D-PDU API/VCI protocol module).

###### H.1.2.4 D-PDU Checker

Checks the validity of PDU data returned by the MSCI protocol module.

###### H.1.2.5 Data Extractor

Extracts the desired information from the PDU data, converts the data to appropriate units and passes the information along to the application.

#### H.1.3 VCI protocol module

##### H.1.3.1 D-PDU API

The D-PDU API processor provides the link between the MSCI protocol module and the MSCI D-Server (or application). It processes all of the function calls received from the D-Server and distributes them to the appropriate processing module. It is also responsible for passing the appropriate responses back to the D-Server.

##### H.1.3.2 D-PDU Scheduler

The D-PDU Scheduler controls when Send ComPrimitives are queued up for transmission via the ComLogicalLink. For cyclic Send ComPrimitives, the Scheduler is responsible for restarting a timer after the transmission has been queued and for keeping track of the number of send cycles that have been completed.

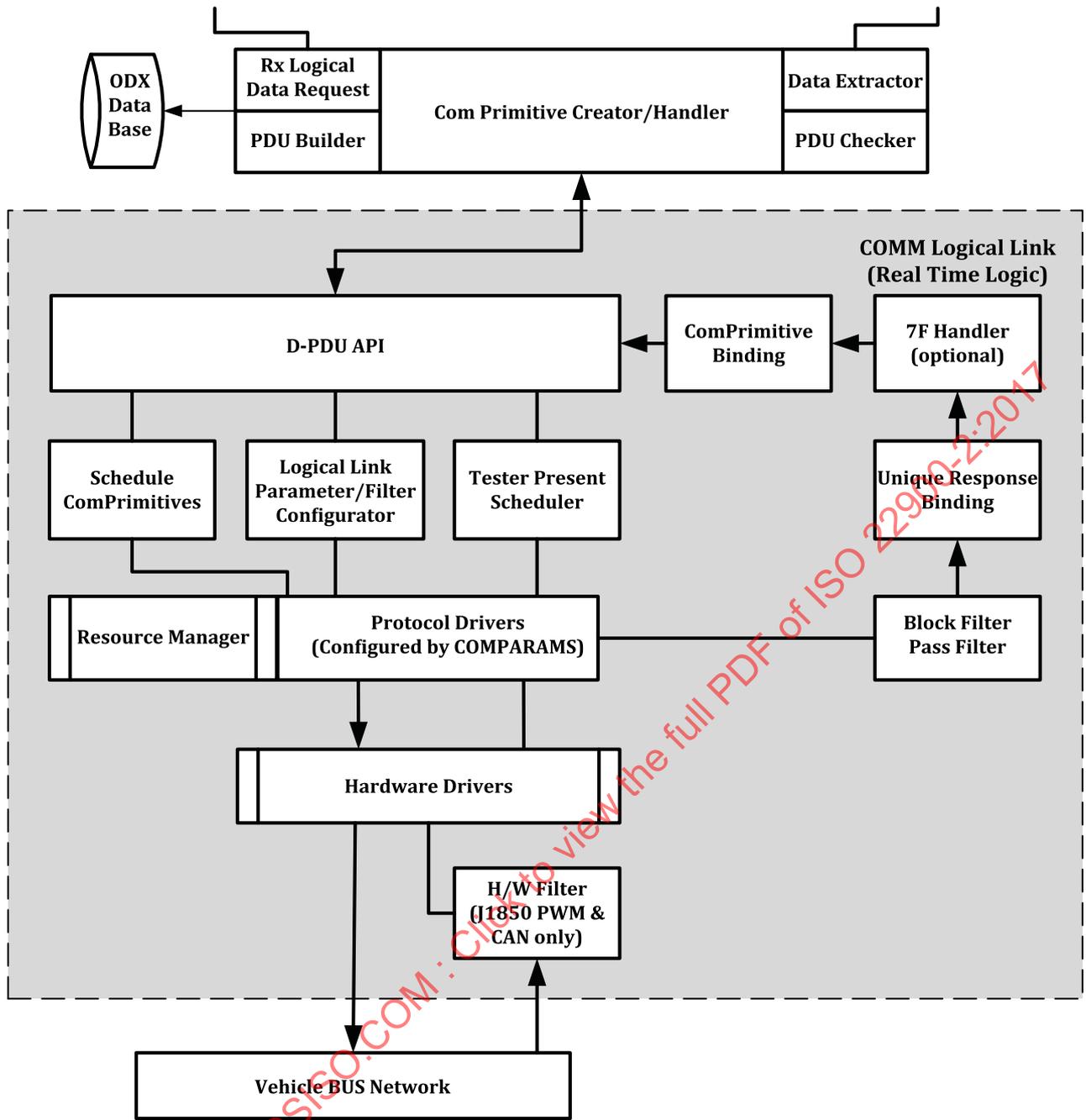


Figure H.1 — Modular VCI protocol module functional partitioning diagram

**H.1.3.3 Logical Link parameter/filter configuration**

Handles configuring a ComLogicalLink and all associated resources assigned to the Logical Link.

**H.1.3.4 Tester present scheduler**

This scheduler controls the transmission of a Tester Present message for protocols that require this functionality. This scheduler controls what is transmitted on the vehicle bus and when it is transmitted.

**H.1.3.5 Resource manager**

Manages the resources on the physical module.

**H.1.3.6 Protocol driver**

Protocol specific driver. (There are nine currently defined for the D-PDU API.) Handle specific timing and formatting requirements for a protocol implementation.

**H.1.3.7 Hardware driver**

Provides the firmware interface to the MVCI protocol module's hardware devices (e.g. UARTs and CAN controllers).

**H.1.3.8 Hardware filters**

Some types of controllers (e.g. CAN controllers and the PWM version of the SAE J1850 interface) have filter capability built into them. Other protocols require that filtering be handled by the MVCI protocol module's firmware (see H.1.3.9).

**H.1.3.9 Software filters****H.1.3.9.1 Block filter**

Messages that are accepted by this filter are discarded.

**H.1.3.9.2 PASS filter**

Messages that are accepted by this filter are passed on.

**H.1.3.10 ISO 15765 USDT/UUDT Frames**

Each USDT CAN Frame for ISO\_15765 protocol shall have a matching entry in the UniqueRespIdTable to be handled in the transport layer. If the frame is USDT and a first frame, then this table is used to send out the correct Flow Control frame. If the CAN ID is a UUDT type of frame or is not in the table, then the message is accepted without any further format checking. The frame data is then checked against the ExpectedResponseStructure to bind the frame to a ComPrimitive.

**H.1.3.11 Negative response Code 0x7F filter handler**

If Negative Response Handling is enabled, each valid message/frame received is checked for a negative response service ID (0x7F) and a known response code (0x21, 0x23, 0x78). If there is a match to the response code, then the proper re-transmission or new receive time handling is started.

**H.1.3.12 Unique response binding**

Match the received message header information to an entry in the table of Unique Response Ids. The matching algorithm is protocol specific (e.g. some protocols will use CAN Ids, others will use Target Addresses, ECU Sources address, etc.).

**H.1.3.13 ComPrimitive binding**

Once a UniqueRespIdentifier is found, the payload data is attempted to be matched to the ExpectedResponseStructure (see 10.1.4.19) of all active ComPrimitives (starting with the active SENDRECV ComPrimitive).

**H.1.4 Vehicle bus network**

The MVCI protocol module interfaces to the vehicle's ECUs via the vehicle bus network. The MVCI protocol module accesses this network via the Data Link Connector (DLC) as described in the ODX Data Base.

## Annex I (informative)

### Use case scenarios

#### I.1 Negative response handling scenarios

##### I.1.1 General

This annex covers the special handling of the Negative Response Codes 0x21, 0x23 and 0x78 for diagnostic protocols such as ISO\_14230\_4 or ISO\_14229\_3.

NOTE Not all of the Negative Response Codes are defined for each protocol.

The processing of handling Negative Response Codes is mainly determined by the CP\_RCxxHandling ComParams.

If Negative Response Codes are received in other cases than specified here, they are simply reported to the application as a ResponseItem. The same applies if any of the Negative Response Codes 0x21, 0x23 or 0x78 are received even though the respective handling ComParam CP\_RCxxHandling does not allow usage of the response code.

In the given figures, the handling of the Negative Response Codes 0x21 and 0x23 is identical. Therefore, only the handling of NRC 0x21 is presented.

In cases where an errorItem (PDU\_ERR\_EVT\_RX\_TIMEOUT) is sent back to the client application, an additional PDU\_XTRA\_ERR... error code may be supplied to give detailed information about the nature of the timeout event.

##### I.1.2 Physical addressing

Figure I.1 shows the processing performed when Negative Error Code 0x21 or 0x23 or 0x78 is received with CP\_RCXXHandling = 0 (disabled). It is up to the client application to handle the negative responses from an ECU. For a negative response RC78, the client application would have to have specified a receive only ComPrimitive to bind the eventual positive response from the ECU.

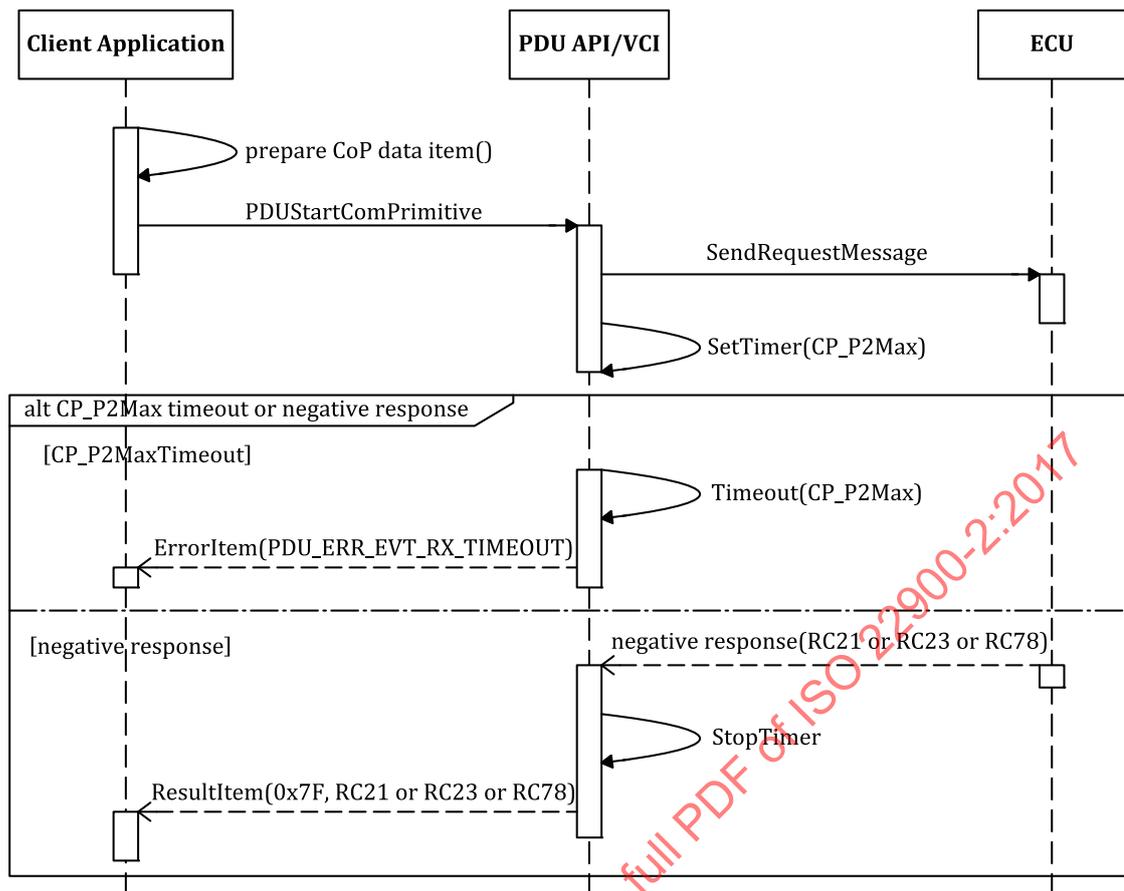


Figure I.1 — Response handling for RC21/RC23, CP\_RCXXHandling = 0

Figure I.2 shows the processing performed when Negative Error Code 0x21 or 0x23 is received with CP\_RCXXHandling = 1 (continue handling negative responses until RCXX\_CompletionTimeout). The ECU is too busy to perform the request and the request is not started. Re-requests are continued until the timeout occurs or until a positive response is received.

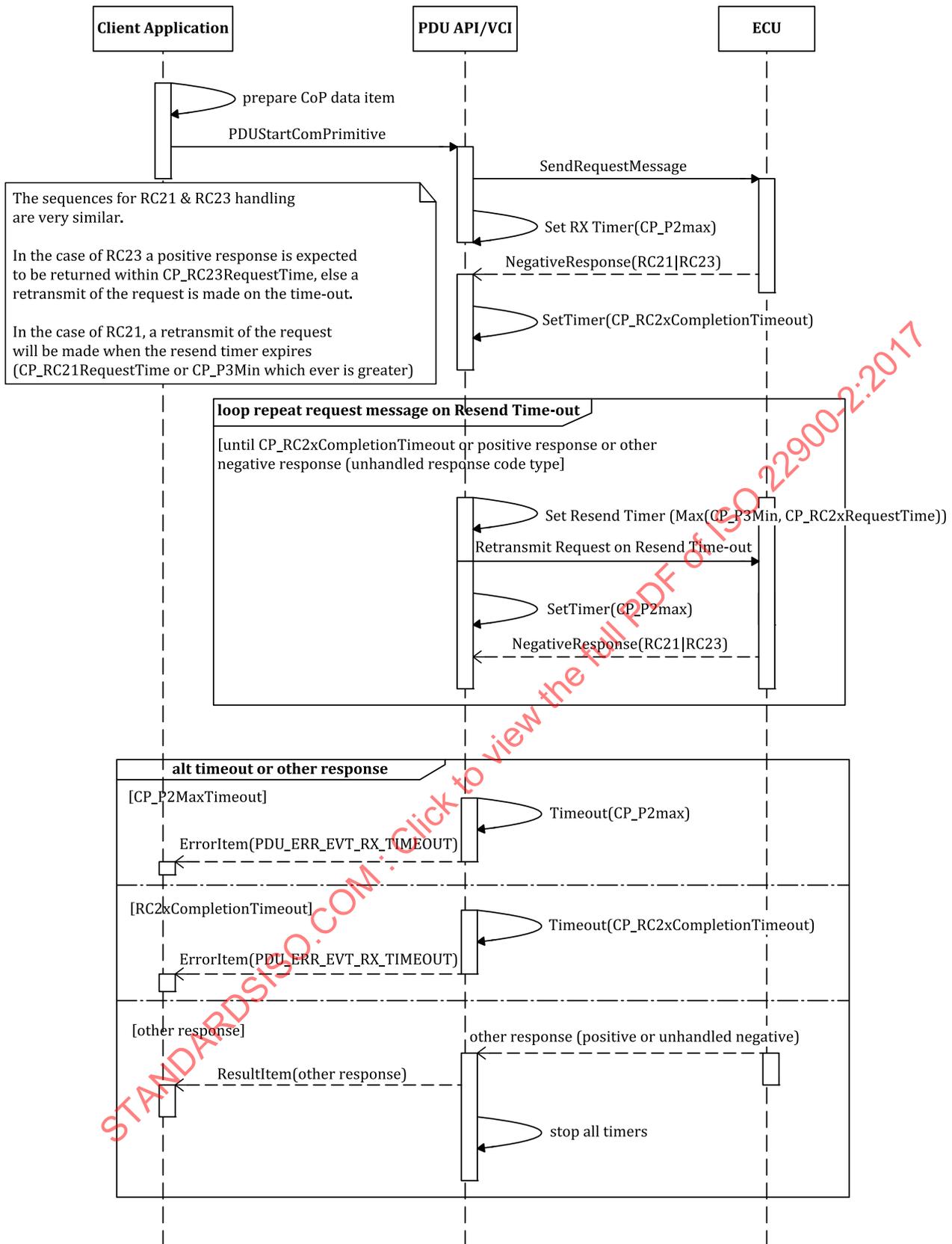


Figure I.2 — Response handling for RC21, RC23, CP\_RCXXHandling = 1

Figure I.3 shows the processing performed when Negative Error Code 0x21 or 0x23 is received with CP\_RCXXHandling = 2 (repeat unlimited). The ECU is too busy to perform the request. The request message is resent until a timeout occurs or a non-error response is received.

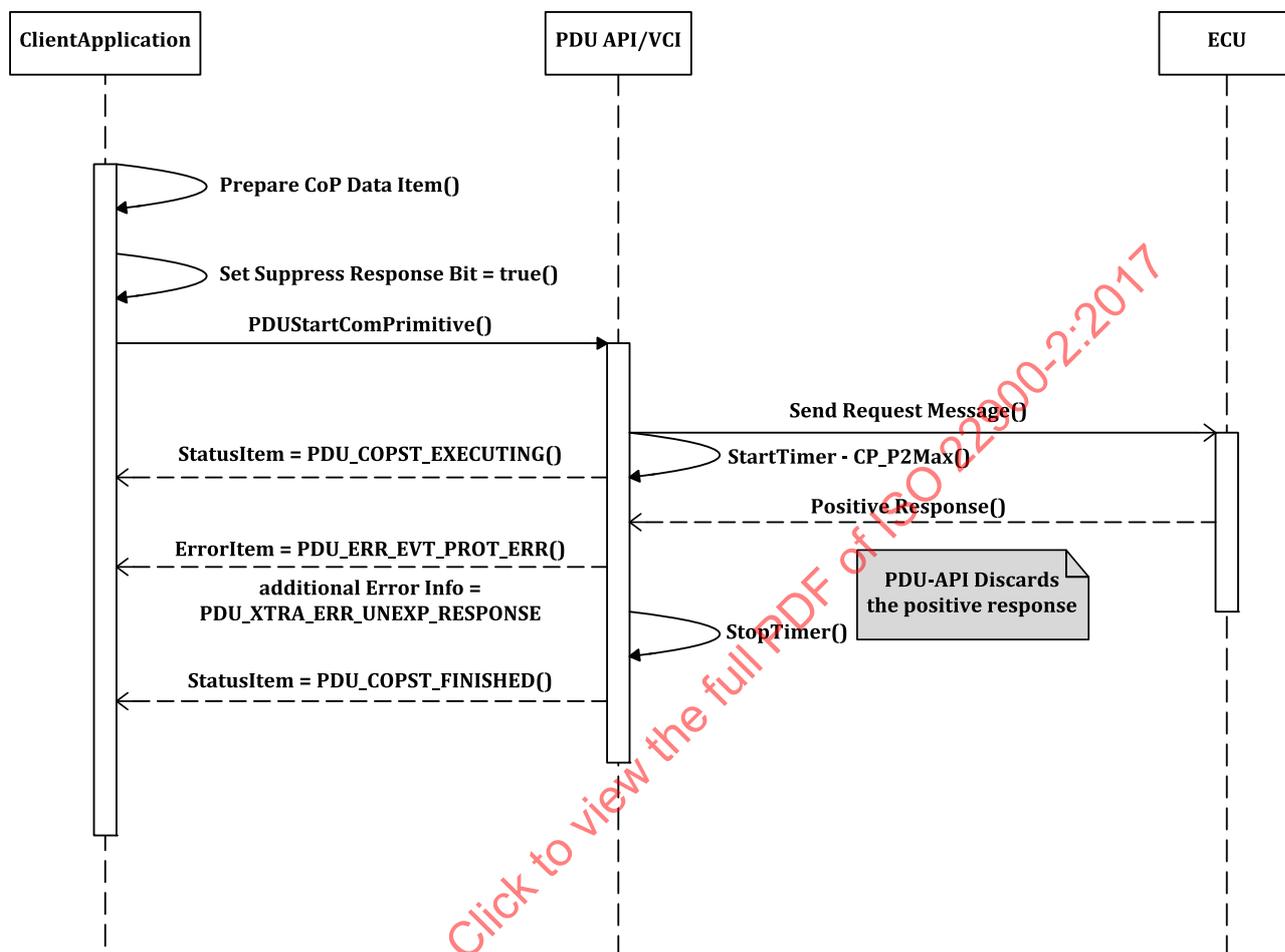


Figure I.3 — Response handling for RC21, RC23, CP\_RCXXHandling = 2