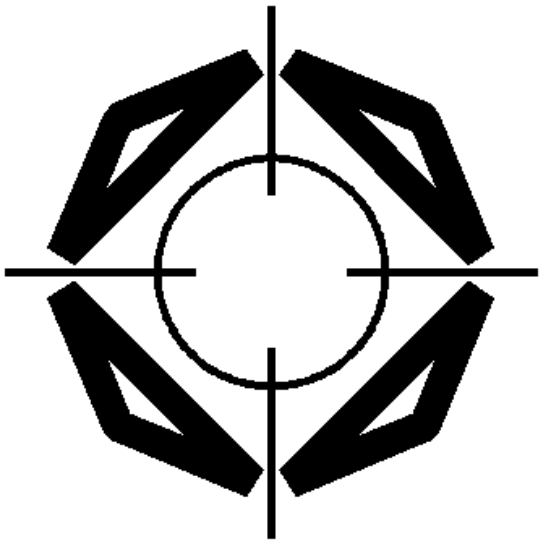
XCP

Version 1.0

“The Universal Measurement and Calibration Protocol Family”

Part 2

Protocol Layer Specification



Association for Standardization of Automation and Measuring Systems

Dated:2003-04-08

Status of Document

Date:	2003-04-08
Authors:	Roel Schuermans, Vector Informatik GmbH Rainer Zaiser, Vector Informatik GmbH Frank Hepperle, DaimlerChrysler AG Hans Schröter, DaimlerChrysler AG Reiner Motz, Robert Bosch GmbH Andreas Aberfeld, Robert Bosch GmbH Hans-Georg Kunz, Siemens VDO Automotive AG Thomas Tyl, Siemens VDO Automotive AG Robert Leinfellner, dSPACE GmbH Hendirk Amsbeck, dSPACE GmbH Harald Styrsky, Compact Dynamics GmbH Boris Ruoff, ETAS GmbH Lars Wahlmann, Accurate Technologies Inc.
Version:	1.0
Doc-ID:	XCP -Part 2- Protocol Layer Specification -1.0
Status:	Released
Type	Final

Revision History

This revision history shows only major modifications between release versions.

Date	Author	Filename	Comments
2003-04-08	R.Schuermans		Released document

Table of contents

0 Introduction 8

0.1 The *X*CP Protocol Family 8

0.2 Documentation Overview 9

0.3 Definitions and Abbreviations 10

1 The *X*CP Protocol Layer 11

1.1 The *X*CP Packet 11
1.1.1 The *X*CP Packet Types 11
1.1.2 The *X*CP Packet Format 12
1.1.2.1 The Identification Field 13
1.1.2.2 The Timestamp Field 18
1.1.2.3 The Data Field 20
1.1.3 The CTO Packets 21
1.1.3.1 Command Packet (CMD) 21
1.1.3.2 Command Response packet (RES) 21
1.1.3.3 Error packet (ERR) 22
1.1.3.4 Event packet (EV) 22
1.1.3.5 Service Request packet (SERV) 22
1.1.4 The DTO Packets 23
1.1.4.1 Data Acquisition Packet (DAQ) 23
1.1.4.2 Synchronous Data Stimulation Packet (STIM) 23
1.1.5 The *X*CP Packet Identifiers 24
1.1.5.1 Master → Slave 24
1.1.5.2 Slave →Master 24

1.2 Table of Event codes (EV) 25

1.3 Table of Service Request codes (SERV) 26

1.4 Table of Command codes (CMD) 27
1.4.1 Standard commands (STD) 28
1.4.2 Calibration commands (CAL) 29
1.4.3 Page switching commands (PAG) 30
1.4.4 Data Acquisition and Stimulation commands (DAQ) 31
1.4.5 Non-volatile memory programming commands (PGM) 32

1.5 Table of bit mask coded parameters 33

1.6 Description of Commands 43
1.6.1 Standard commands (STD) 44
1.6.1.1 Mandatory commands 44
1.6.1.1.1 Set up connection with slave 44
1.6.1.1.2 Disconnect from slave 47
1.6.1.1.3 Get current session status from slave 48
1.6.1.1.4 Synchronize command execution after time-out 53
1.6.1.2 Optional commands 54
1.6.1.2.1 Get communication mode info 54
1.6.1.2.2 Get identification from slave 56

1.6.1.2.3 Request to save to non-volatile memory 57
1.6.1.2.4 Get seed for unlocking a protected resource 58
1.6.1.2.5 Send key for unlocking a protected resource 59
1.6.1.2.6 Set Memory Transfer Address in slave 62
1.6.1.2.7 Upload from slave to master 63
1.6.1.2.8 Upload from slave to master (short version) 65
1.6.1.2.9 Build checksum over memory range 66
1.6.1.3 Auxiliary commands 68
1.6.1.3.1 Refer to transport layer specific command 68
1.6.1.3.2 Refer to user defined command 69
1.6.2 Calibration commands (CAL) 70
1.6.2.1 Mandatory commands 70
1.6.2.1.1 Download from master to slave 70
1.6.2.2 Optional commands 71
1.6.2.2.1 Download from master to slave (Block Mode) 71
1.6.2.2.2 Download from master to slave (fixed size) 72
1.6.2.2.3 Download from master to slave (short version) 73
1.6.2.2.4 Modify bits 74
1.6.3 Page switching commands (PAG) 75
1.6.3.1 Mandatory commands 75
1.6.3.1.1 Set calibration page 75
1.6.3.1.2 Get calibration page 76
1.6.3.2 Optional commands 77
1.6.3.2.1 Get general information on PAG processor 77
1.6.3.2.2 Get specific information for a SEGMENT 78
1.6.3.2.3 Get specific information for a PAGE 80
1.6.3.2.4 Set mode for a SEGMENT 83
1.6.3.2.5 Get mode for a SEGMENT 84
1.6.3.2.6 Copy page 85
1.6.4 Data Acquisition and Stimulation Commands (DAQ) 86
1.6.4.1 Static DAQ list configuration (stat) 86
1.6.4.1.1 Mandatory commands 86
1.6.4.1.1.1 Clear DAQ list configuration 86
1.6.4.1.1.2 Set pointer to ODT entry 87
1.6.4.1.1.3 Write element in ODT entry 88
1.6.4.1.1.4 Set mode for DAQ list 89
1.6.4.1.1.5 Get mode from DAQ list 91
1.6.4.1.1.6 Start /stop/select DAQ list 93
1.6.4.1.1.7 Start/stop DAQ lists (synchronously) 94
1.6.4.1.2 Optional commands 95
1.6.4.1.2.1 Get DAQ clock from slave 95
1.6.4.1.2.2 Read element from ODT entry 96
1.6.4.1.2.3 Get general information on DAQ processor 97
1.6.4.1.2.4 Get general information on DAQ processing resolution 101
1.6.4.1.2.5 Get specific information for a DAQ list 103
1.6.4.1.2.6 Get specific information for an event channel 105
1.6.4.2 Dynamic DAQ List Configuration (dyn) 107
1.6.4.2.1 Optional commands 107
1.6.4.2.1.1 Clear dynamic DAQ configuration 107
1.6.4.2.1.2 Allocate DAQ lists 108

1.6.4.2.1.3 Allocate ODTs to a DAQ list ... 109
1.6.4.2.1.4 Allocate ODT entries to an ODT ... 110
1.6.5 Non-volatile Memory Programming (PGM) ... 111
1.6.5.1 Mandatory commands ... 111
1.6.5.1.1 Indicate the beginning of a programming sequence ... 111
1.6.5.1.2 Clear a part of non-volatile memory ... 113
1.6.5.1.3 Program a non-volatile memory segment ... 115
1.6.5.1.4 Indicate the end of a programming sequence ... 116
1.6.5.2 Optional commands ... 117
1.6.5.2.1 Get general information on PGM processor ... 117
1.6.5.2.2 Get specific information for a SECTOR ... 120
1.6.5.2.3 Prepare non-volatile memory programming ... 121
1.6.5.2.4 Set data format before programming ... 122
1.6.5.2.5 Program a non-volatile memory segment (Block Mode) ... 124
1.6.5.2.6 Program a non-volatile memory segment (fixed size) ... 125
1.6.5.2.7 Program Verify ... 126
1.7 Communication Error Handling ... 127
1.7.1 Definitions ... 127
1.7.1.1 Error ... 127
1.7.1.2 Pre-action ... 128
1.7.1.3 Action ... 129
1.7.1.4 Error severity ... 130
1.7.2 Time-out Handling ... 131
1.7.2.1 Standard Communication Model ... 131
1.7.2.2 Block Communication Model ... 132
1.7.2.3 Interleaved Communication Model ... 133
1.7.2.4 Time-Out manipulation ... 134
1.7.2.4.1 Overruling Time-Out values ... 134
1.7.2.4.2 Restarting Time-Out detection ... 135
1.7.3 Error Code Handling ... 136
1.7.3.1 Table of Error Codes (ERR_*) ... 137
1.7.3.2 The Error Handling Matrix ... 138
1.7.3.2.1 Standard commands (STD) ... 138
1.7.3.2.2 Calibration commands (CAL) ... 140
1.7.3.2.3 Page switching commands (PAG) ... 141
1.7.3.2.4 Data Acquisition and Stimulation commands (DAQ) ... 142
1.7.3.2.5 Non-volatile memory programming commands (PGM) ... 144
2 Interface to ASAM MCD 2MC description file ... 146
2.1 ASAM MCD 2MC AML for *X*CP (Common_Parameters) ... 146

Table of diagrams :

Diagram 1 : Communication flow between master and slave devices 11
Diagram 2 : The *X*CP Packet format 12
Diagram 3 : The *X*CP Packet Identification Field 13
Diagram 4 : Identification Field Type "CTO Packet Code" 13
Diagram 5 : Identification Field Type "absolute ODT number" 14
Diagram 6 : Identification Field Type "relative ODT number and absolute DAQ list number (BYTE)" 15
Diagram 7 : Identification Field Type "relative ODT number and absolute DAQ list number (WORD)" 15
Diagram 8 : Identification Field Type "relative ODT number and absolute DAQ list number (WORD, aligned)" 16
Diagram 9 : The *X*CP Packet Timestamp Field 18
Diagram 10 : TS only in first DTO Packet of sample 18
Diagram 11 : Timestamp Field Types 19
Diagram 12 : The *X*CP Packet Data Field 20
Diagram 13 : The CTO Packet 21
Diagram 14 : The DTO Packet 23
Diagram 15 : The *X*CP Packet IDentifiers from Master to Slave 24
Diagram 16 : The *X*CP Packet IDentifiers from Slave to Master 24
Diagram 17 : short GET_SEED+UNLOCK sequence 60
Diagram 18 : long GET_SEED+UNLOCK sequence 61
Diagram 19 : UPLOAD 3 bytes 64
Diagram 20 : UPLOAD 7 bytes 64
Diagram 21 : UPLOAD 16 bytes in block mode 64
Diagram 22 : DOWNLOAD 6 bytes 70
Diagram 23 : DOWNLOAD 14 bytes in block mode 71
Diagram 24 : Time-out Handling in Standard Communication Model 131
Diagram 25 : Time-out Handling in Master Block Transfer Mode 132
Diagram 26 : Time-out Handling in Slave Block Transfer Mode 132
Diagram 27 : Time-out Handling in Interleaved Communication Model 133
Diagram 28 : Restarting Time-out detection with EV_CMD_PENDING 135

0 Introduction

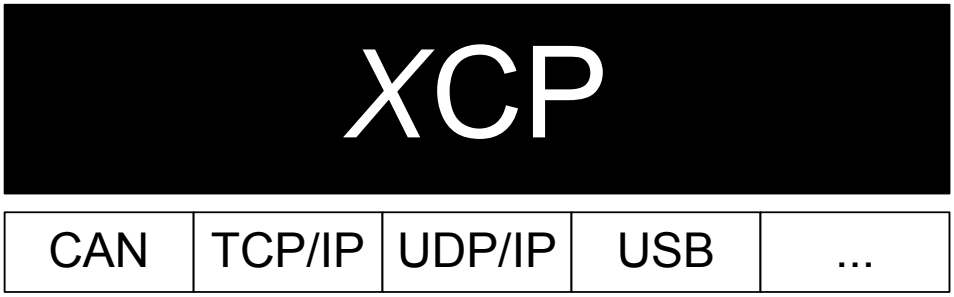
0.1 The *X*CP Protocol Family

This document is based on experiences with the **C**AN **C**alibration **P**rotocol (CCP) version 2.1 as described in feedback from the companies Accurate Technologies Inc., Compact Dynamics GmbH, DaimlerChrysler AG, dSPACE GmbH, ETAS GmbH, Kleinknecht Automotive GmbH, Robert Bosch GmbH, Siemens VDO Automotive AG and Vector Informatik GmbH.

The *X*CP Specification documents describe an improved and generalized version of CCP.

The generalized protocol definition serves as standard for a protocol family and is called "*X*CP" (Universal Measurement and **C**alibration **P**rotocol).

The **"*X*"** generalizes the "various" transportation layers that are used by the members of the protocol family e.g "*X*CP on CAN", "*X*CP on TCP/IP", "*X*CP on UDP/IP", "*X*CP on USB" and so on.



0.2 Documentation Overview

The *X*CP specification consists of 5 parts. Each part is a separate document and has the following contents:

Part 1 "Overview" gives an overview over the *X*CP protocol family, the *X*CP features and the fundamental protocol definitions.

Part 2 "Protocol Layer Specification" defines the generic protocol, which is independent from the transportation layer used (this document).

Part 3 "Transport Layer Specification" defines the way how the *X*CP protocol is transported by a particular transportation layer like CAN, TCP/IP and UDP/IP.

Part 4 "Interface Specification" defines the interfaces from an *X*CP master to an ASAM MCD 2MC description file and for calculating Seed & Key algorithms and checksums.

Part 5 "Example Communication Sequences" gives example sequences for typical actions performed with *X*CP.

Everything not explicitly mentioned in this document, should be considered as implementation specific.

0.3 Definitions and Abbreviations

The following table gives an overview about the most commonly used definitions and abbreviations throughout this document.

Abbreviation	Description
A2L	File Extension for an **A**SAM **2**MC **L**anguage File
AML	**A**SAM 2 **M**eta **L**anguage
ASAM	**A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems
BYP	**BYP**assing
CAL	**CAL**ibration
CAN	**C**ontroller **A**rea **N**etwork
CCP	**C**an **C**alibration **P**rotocol
CMD	**C**o**M**man**D**
CS	**C**heck**S**um
CTO	**C**ommand **T**ransfer **O**bject
CTR	**C**oun**T**e**R**
DAQ	**D**ata **A**c**Q**uisition, **D**ata **A**c**Q**uisition Packet
DTO	**D**ata **T**ransfer **O**bject
ECU	**E**lectronic **C**ontrol **U**nit
ERR	**ERR**or Packet
EV	**EV**ent Packet
LEN	**LEN**gth
MCD	**M**easurement **C**alibration and **D**iagnostics
MTA	**M**emory **T**ransfer **A**ddress
ODT	**O**bject **D**escriptor **T**able
PAG	**PAG**ing
PGM	**P**ro**G**ra**M**ming
PID	**P**acket **ID**entifier
RES	command **RES**ponse packet
SERV	**SERV**ice request packet
SPI	**S**erial **P**eripheral **I**nterface
STD	**ST**an**D**ard
STIM	Data **STIM**ulation packet
TCP/IP	**T**ransfer **C**ontrol **P**rotocol **/ I**nternet **P**rotocol
TS	**T**ime Stamp
UDP/IP	**U**nified **D**ata **P**rotocol **/ I**nternet **P**rotocol
USB	**U**niversal **S**erial **B**us
XCP	Universal **C**alibration **P**rotocol

Table 1: Definitions and Abbreviations

1 The *X*CP Protocol Layer

1.1 The *X*CP Packet

1.1.1 The *X*CP Packet Types

All *X*CP communication is transferred as data objects called *X*CP Packets.

There are 2 basic Packet types:

1) Packet for transferring generic control commands : **CTO**
2) Packet for transferring synchronous data : **DTO**

The **CTO** (**C**ommand **T**ransfer **O**bject) is used for transferring generic control commands. It is used for carrying out protocol commands (CMD), transferring command responses (RES), error (ERR) packets, event (EV) packets and for service request packets (SERV).

The **DTO** (**D**ata **T**ransfer **O**bject) is used for transmitting synchronous data acquisition data (DAQ) and for transmitting synchronous data stimulation data (STIM).

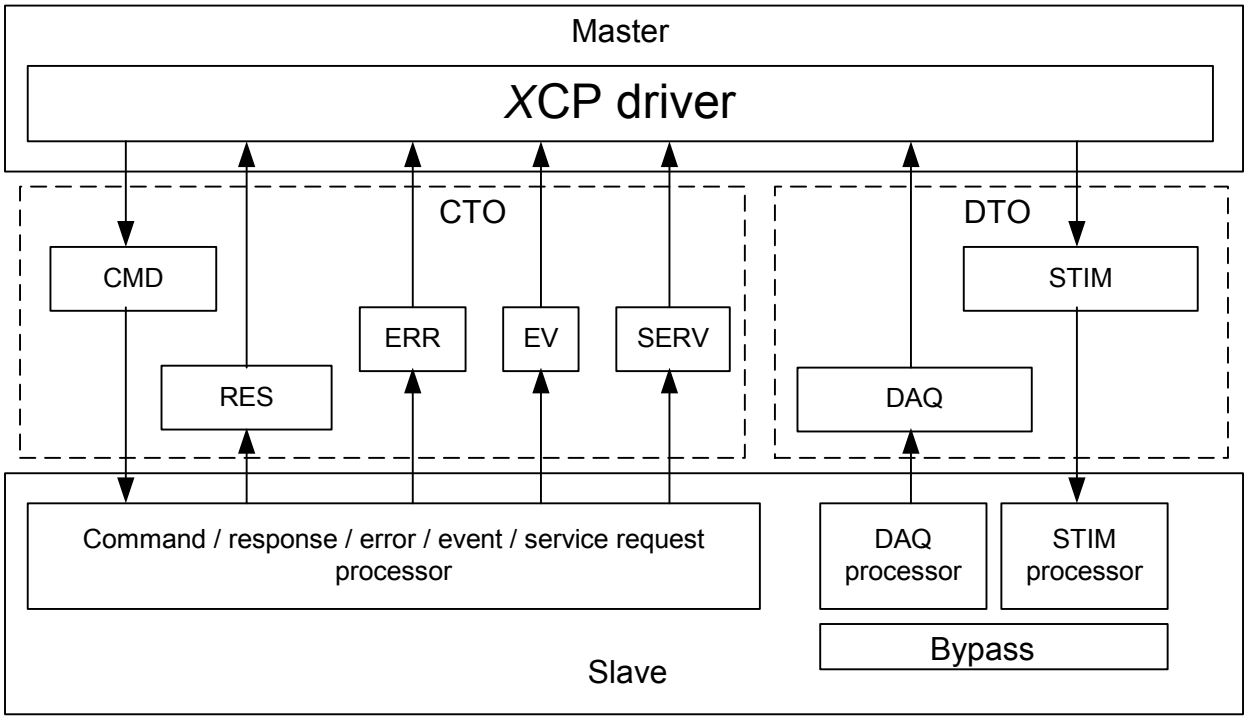


Diagram 1 : Communication flow between master and slave devices

A Command Packet must always be answered by a Command Response Packet or an Error Packet.

Event, Service Request and Data Acquisition Packets are send asynchronously, therefore it may not be guaranteed that the master device will receive them when using a non acknowledged transportation link like e.g. UDP/IP.

1.1.2 The *X*CP Packet Format

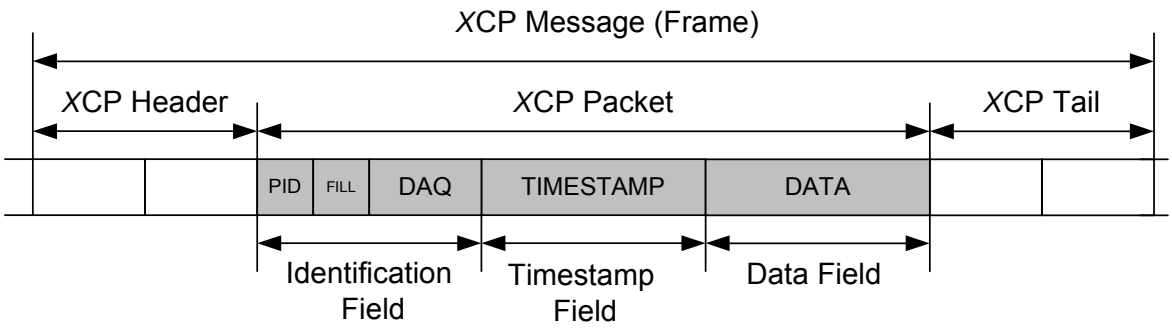


Diagram 2 : The *X*CP Packet format

The *X*CP Packet contains the generic part of the protocol, which is independent from the transport layer used.

An *X*CP Packet consists of an Identification Field, an optional Timestamp Field and a Data Field.

MAX_CTO indicates the maximum length of a CTO packet in bytes.
MAX_DTO indicates the maximum length of a DTO packet in bytes.

1.1.2.1 The Identification Field

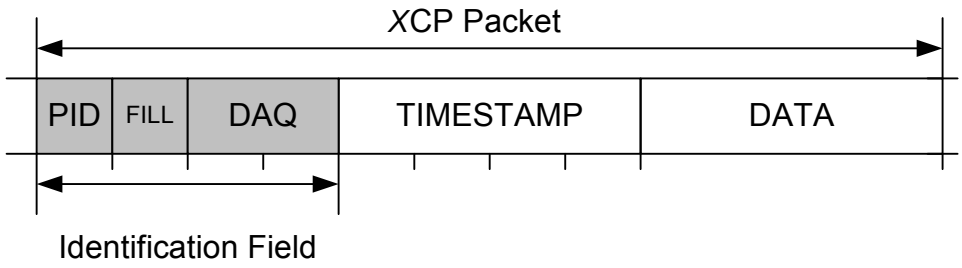


Diagram 3 : The *X*CP Packet Identification Field

When exchanging *X*CP Packets, both master and slave always have to be able to unambiguously identify any transferred *X*CP Packet concerning its Type and the contents of its Data Field.

For this purpose, an *X*CP Packet basically always starts with an Identification Field which as first byte contains the **P**acket **ID**entifier (PID).

Identification Field Type "CTO Packet Code"

For CTO Packets, the Identification Field should be able to identify the packets concerning their Type, distinguishing between protocol commands (CMD), command responses (RES), error packets (ERR), event packets (EV) and service request packets (SERV).

For CTO Packets, the Identification Field just consists of the PID, containing the CTO Packet code.

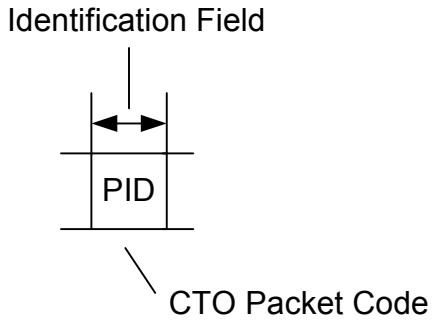


Diagram 4 : Identification Field Type "CTO Packet Code"

For DTO Packets, the Identification Field should be able to identify the packets concerning their Type, distinguishing between DTO Packets for Synchronous Data Acquisition or for Synchronous Data Stimulation

For DTO Packets, the Identification Field should be able to identify unambiguously the DAQ list and the ODT within this DAQ list, that describe the contents of the Data Field.

For every DAQ list the numbering of the ODTs through ODT_NUMBER restarts from 0:

DAQ list 0	DAQ list 1	…
ODT 0	ODT **0**	...
ODT 1	...	

so the scope for ODT_NUMBER is local for a DAQ list and ODT numbers are not unique within one and the same slave device.

Identification Field Type "absolute ODT number"

One possibility to map the relative and not unique ODT numbers to unambiguously identifiable DTO Packets, is to map the relative ODT numbers to absolute ODT numbers by means of a "FIRST_PID for this DAQ list", and then transfer the absolute ODT numbers within the DTO Packet.

The following mapping from relative_ODT_NUMBER to absolute_ODT_NUMBER applies:

absolute_ODT_NUMBER(ODT i in DAQ list j)
= FIRST_PID(DAQ list j) + relative_ODT_NUMBER(ODT i)

FIRST_PID is the PID in the DTO Packet of the first ODT transferred by this DAQ list.

FIRST_PID is determined by the slave device and sent to the master upon START_STOP_DAQ_LIST(DAQ list j).

When allocating the FIRST_PIDs, the slave has to make sure that for every ODT there's a unique absolute ODT number.

All PIDs also have to be in the available ranges for PID(DAQ) and PID(STIM).

For DTO Packets with Identification Field Type "absolute ODT number", the Identification Field just consists of the PID, containing the absolute ODT number.

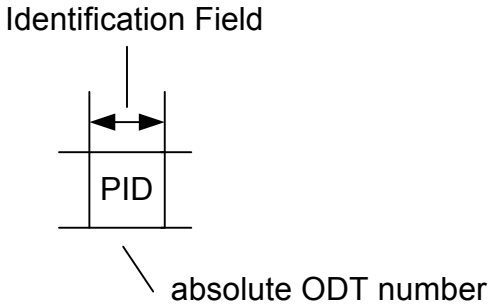


Diagram 5 : Identification Field Type "absolute ODT number"

Identification Field Type "relative ODT number and absolute DAQ list number"

Another possibility to map the relative and not unique ODT numbers to unambiguously identifiable DTO Packets, is to transfer the absolute DAQ list number together with the relative ODT number within the DTO Packet.

For DTO Packets with Identification Field Types "relative ODT number and absolute DAQ list number", the Identification Field consists of the PID, containing the relative ODT number, DAQ bits, containing the absolute DAQ list number, and an optional FILL byte.

One possibility is to transfer the DAQ list number as BYTE, which reduces the number of theoretically possible Packets since the DAQ_LIST_NUMBER parameter is coded as WORD.

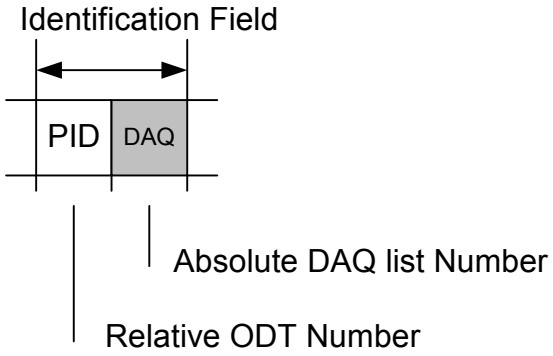


Diagram 6 : Identification Field Type "relative ODT number and absolute DAQ list number (BYTE)"

For fully exploring the limits of performance, there's the possibility to transfer the DAQ list number as WORD

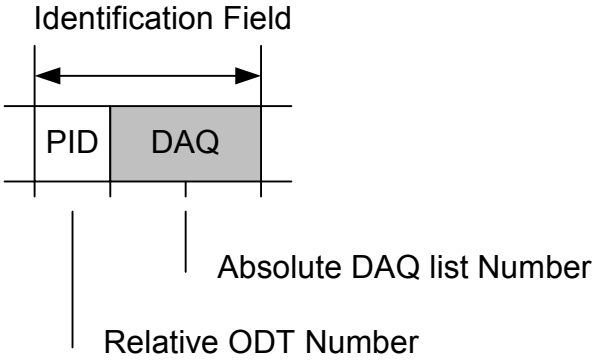


Diagram 7 : Identification Field Type "relative ODT number and absolute DAQ list number (WORD)"

If for the *X*CP Packet certain alignment conditions have to be met, there's the possibility to transfer an extra FILL byte.

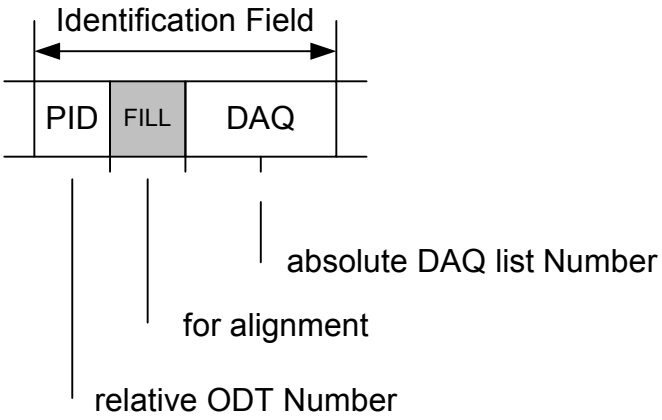


Diagram 8 : Identification Field Type
"relative ODT number and absolute DAQ list number (WORD, aligned)"

With the DAQ_KEY_BYTE at GET_DAQ_PROCESSOR_INFO, the slave informs the master about the Type of Identification Field the slave will use when transferring DAQ Packets to the master. The master has to use the same Type of Identification Field when transferring STIM Packets to the slave.

Empty Identification Field

A DAQ list can have the property that it can transmit DTO Packets without Identification Field (ref. PID_OFF_SUPPORTED flag in DAQ_PROPERTIES at GET_DAQ_PROCESSOR_INFO).

Turning off the transmission of the Identification Field is only allowed if the Identification Field Type is “absolute ODT number”. If the Identification Field is not transferred in the *X*CP Packet, the unambiguous identification has to be done on the level of the Transport Layer. This can be done e.g. on CAN with separate CAN-Ids for each DAQ list and only one ODT for each DAQ list. In this case turning off the Identification Field would allow the transmission of 8 byte signals on CAN.

1.1.2.2 The Timestamp Field

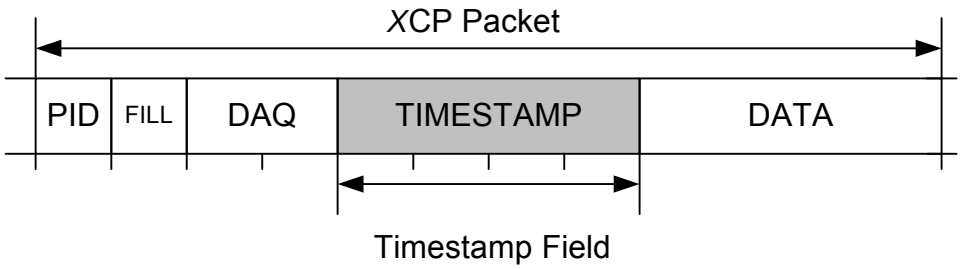


Diagram 9 : The *X*CP Packet Timestamp Field

An *X*CP Packet optionally might contain a Timestamp Field.

For CTO Packets, the Timestamp Field is not available.
DTO Packets directly after the Identification Field might have a Timestamp Field which contains a **T**ime**S**tamp (TS).

The TIMESTAMP_SUPPORTED flag at GET_DAQ_PROCESSOR_INFO indicates whether the slave supports time stamped data acquisition and stimulation.

With the TIMESTAMP flag at SET_DAQ_LIST_MODE, the master can set a DAQ list into time stamped mode.

The TIMESTAMP_FIXED flag in TIMESTAMP_MODE at GET_DAQ_RESOLUTION_INFO indicates that the Slave always will send DTO Packets in time stamped mode. The Master can not switch off the time stamp with SET_DAQ_LIST_MODE.

The TIMESTAMP flag can be used as well for DIRECTION = DAQ as for DIRECTION = STIM.

For DIRECTION = DAQ, time stamped mode means that the slave device transmits the current value of its clock in the DTO Packet for the first ODT of a DAQ cycle.

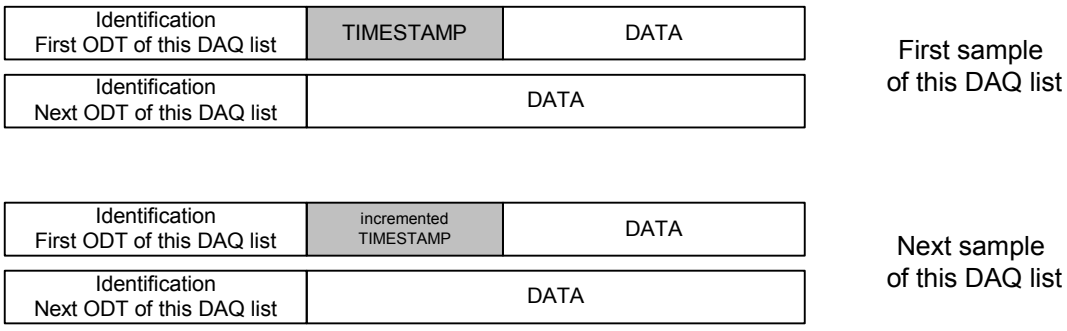


Diagram 10 : TS only in first DTO Packet of sample

For DIRECTION = STIM, time stamped mode means that the master device first receives a time stamped DTO(DAQ) from the slave and then echoes this current value of the slave device's clock in the DTO Packet for the first ODT of the DAQ cycle. In this way the "time stamp" can be used as a counter that gives the slave the possibility to check whether DTO(DAQ) and CTO(STIM) belong functionally together.

Timestamp Field Types

The Timestamp Field always consists of the TS, containing the current value of the synchronous data transfer clock

The synchronous data transfer clock is a free running counter in the slave, which is never reset or modified.

Depending on the Timestamp Field Type, the TS is transferred as BYTE, WORD or DWORD value.

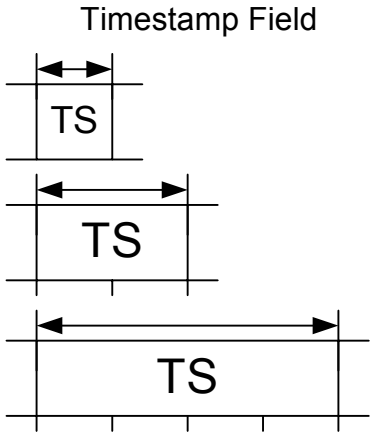


Diagram 11 : Timestamp Field Types

With TIMESTAMP_MODE and TIMESTAMP_TICKS at GET_DAQ_RESOLUTION_INFO, the slave informs the master about the Type of Timestamp Field the slave will use when transferring DAQ Packets to the master. The master has to use the same Type of Timestamp Field when transferring STIM Packets to the slave. TIMESTAMP_MODE and TIMEPSTAMP_TICKS contain information on the resolution of the data transfer clock.

1.1.2.3 The Data Field

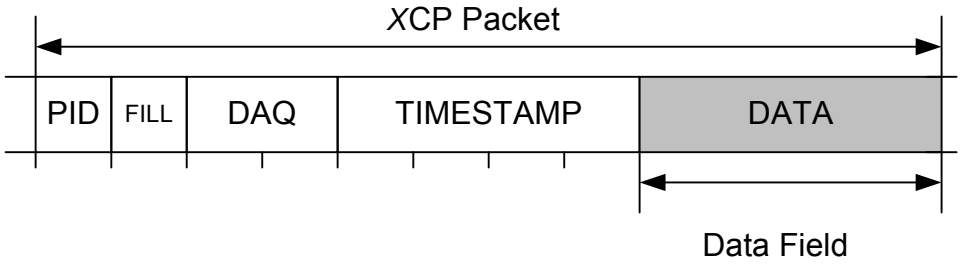


Diagram 12 : The *X*CP Packet Data Field

An *X*CP Packet finally contains a Data Field.

For CTO Packets, the Data Field contains the specific parameters for the different types of CTO packet.

For DTO Packets, the Data Field contains the data for synchronous acquisition and stimulation.

1.1.3 The CTO Packets

The **CTO** is used for transferring generic control commands.

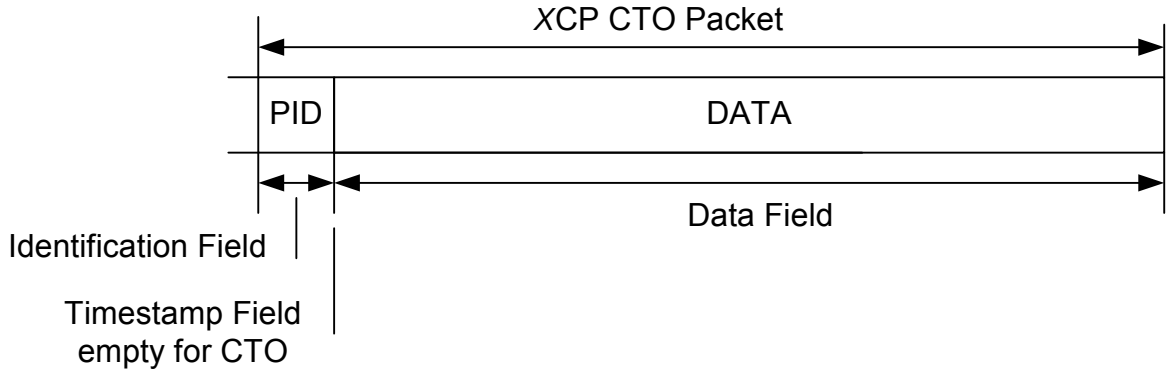


Diagram 13 : The CTO Packet

The Identification Field just consists of the PID, containing the CTO Packet code.
The Timestamp Field is not available.
The Data Field contains the specific parameters for the different types of CTO packet.

1.1.3.1 Command Packet (CMD)

Position	Type	Description
0	BYTE	Packet Identifier = CMD 0xC0...0xFF
1..MAX_CTO-1	BYTE	Command Data

The PID contains the **C**om**M**an**D** Packet code in the range **0xC0 <= CMD <= 0xFF**.
All possible command codes are defined in the section “Table of Command Codes (CMD)” in this paper. The structure of all possible commands is defined in the “Description of Commands” section of this paper.

1.1.3.2 Command Response packet (RES)

Position	Type	Description
0	BYTE	Packet Identifier = RES 0xFF
1..MAX_CTO-1	BYTE	Command response data

The PID contains the Command Positive **RES**ponse Packet code **RES = 0xFF**.
The RES is sent as an answer to a CMD if the command has been successfully executed.

1.1.3.3 Error packet (ERR)

Position	Type	Description
0	BYTE	Packet Identifier = ERR 0xFE
1	BYTE	Error code
2..MAX_CTO-1	BYTE	Optional error information data

The PID contains the **ERR**or Packet code **ERR = 0xFE**.

The ERR is sent as an answer to a CMD if the command has not been successfully executed. The second byte contains the Error code. Error codes are defined in the section “Table of Error codes (ERR_*)” in this paper.

The Error code **0x00** is used for synchronization purposes (ref. description of command SYNCH).
An Error code **ERR_* >= 0x01** is used for Error packets.

Error packets normally only contain an error code.

However, in some cases the error packet contains additional information.

At BUILD_CHECKSUM the error packet with error code 0x22 = ERR_OUT_OF_RANGE contains the maximum allowed block size as DWORD as additional information.

If the error code is 0x31 = ERR_GENERIC, the error packet contains an implementation specific slave device error code as WORD as additional information.

1.1.3.4 Event packet (EV)

Position	Type	Description
0	BYTE	Packet Identifier = EV 0xFD
1	BYTE	Event code
2..MAX_CTO-1	BYTE	Optional event information data

The PID contains the **EV**ent Packet code **EV = 0xFD**.

The EV is sent if the slave wants to report an asynchronous event packet. The second byte contains the Event code. Event codes are defined in the section “Table of Event codes (EV)” in this paper.

The implementation is optional. Event packets sent from the slave device to the master device are not acknowledged, therefore the transmission is not guaranteed.

1.1.3.5 Service Request packet (SERV)

Position	Type	Description
0	BYTE	Packet Identifier = SERV 0xFC
1	BYTE	Service request code
2..MAX_CTO-1	BYTE	Optional service request data

The PID contains the **SERV**ice Request Packet code **SERV = 0xFC**.
The SERV requests some action to be performed by the master device. The second byte contains the service request code. Possible service request codes are defined in the section “Table of Service Request codes” in this paper.

1.1.4 The DTO Packets

The **DTO** is used for transmitting synchronous data acquisition data (DAQ), and for transmitting synchronous data stimulation data (STIM).

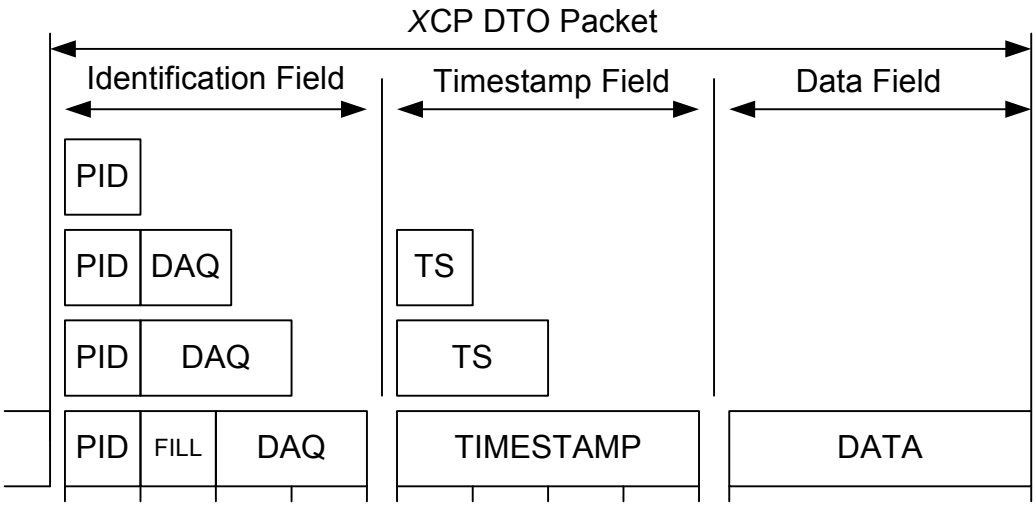


Diagram 14 : The DTO Packet

The contents of the Identification Field varies depending upon the Identification Field Type.
The contents of the Timestamp Field varies depending upon the Timestamp Field Type.
Any combination of Identification Field Type and Timestamp Field Type is possible.
The Data Field contains the data for synchronous acquisition and stimulation.

1.1.4.1 Data Acquisition Packet (DAQ)

Position	Type	Description
0	BYTE	Packet Identifier = DAQ 0x00...0xFB
1..n	BYTE	Rest of Identification Field
n+1..MAX_DTO-1	BYTE	Data

n = f(Identification Field Type, Timestamp Field Type)

The PID contains the (absolute or relative) ODT number in the range **0x00 <= DAQ <= 0xFB**. The ODT number refers to an **O**bject **D**escriptor **T**able (ODT) that describes which data acquisition elements are contained in the remaining data bytes.

1.1.4.2 Synchronous Data Stimulation Packet (STIM)

Position	Type	Description
0	BYTE	Packet Identifier = STIM 0x00...0xBF
1..n	BYTE	Rest of Identification Field
n+1..MAX_DTO-1	BYTE	Data

n = f(Identification Field Type, Timestamp Field Type)

The PID contains the (absolute or relative) ODT number in the range **0x00 <= STIM <= 0xBF**. The ODT number refers to a corresponding **O**bject **D**escriptor **T**able (ODT) that describes which data stimulation elements are contained in the remaining data bytes.

1.1.5 The *X*CP Packet Identifiers

The following tables give an overview of all possible Packet IDentifiers for transferring Packets from Master to Slave and from Slave to Master.

1.1.5.1 Master → Slave

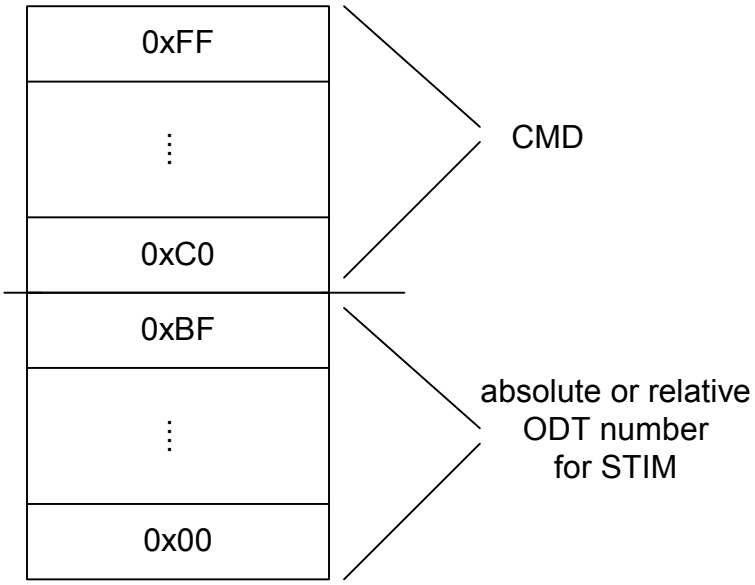


Diagram 15 : The *X*CP Packet IDentifiers from Master to Slave

1.1.5.2 Slave →Master

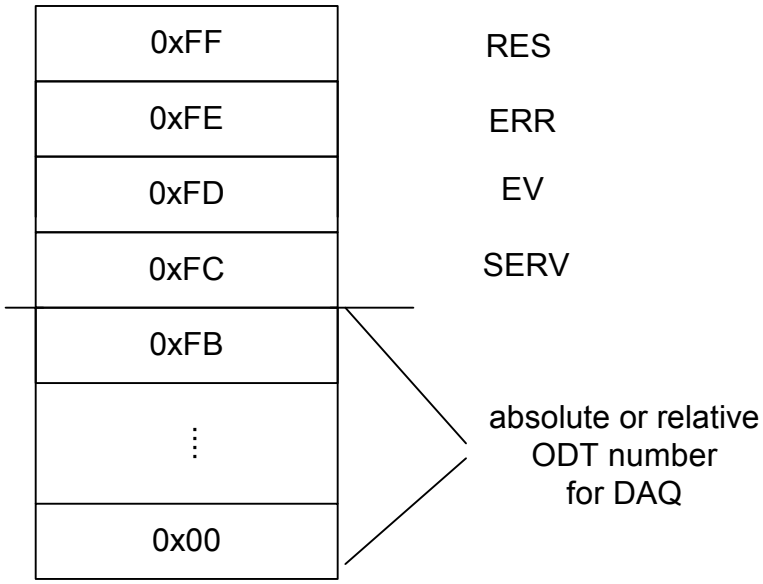


Diagram 16 : The *X*CP Packet IDentifiers from Slave to Master

1.2 Table of Event codes (EV)

The Event packet codes in the table below may be sent as an asynchronous packet with PID 0xFD.

The implementation is optional. Event packets sent from the slave device to the master device are not acknowledged, therefore the transmission is not guaranteed.

Event	Code	Description	Severity
EV_RESUME_MODE	0x00	Slave starting in RESUME mode	S0
EV_CLEAR_DAQ	0x01	The DAQ configuration in non-volatile memory has been cleared.	S0
EV_STORE_DAQ	0x02	The DAQ configuration has been stored into non-volatile memory.	S0
EV_STORE_CAL	0x03	The calibration data has been stored into non-volatile memory.	S0
EV_CMD_PENDING	0x05	Slave requesting to restart time-out	S1
EV_DAQ_OVERLOAD	0x06	DAQ processor overload.	S1
EV_SESSION_TERMINATED	0x07	Session terminated by slave device.	S3
EV_USER	0xFE	User-defined event	S0
EV_TRANSPORT	0xFF	Transport layer specific event	Ref. Part3

With EV_RESUME_MODE the slave indicates that it is starting in RESUME mode.

With EV_CLEAR_DAQ the slave indicates that the DAQ configuration in non-volatile memory has been cleared.

With EV_STORE_DAQ the slave indicates that the DAQ configuration has been stored into non-volatile memory.

With EV_STORE_CAL the slave indicates that calibration data have been stored into non-volatile memory.

With EV_CMD_PENDING the slave requests the master to restart the time-out detection.

With EV_DAQ_OVERLOAD the slave may indicate an overload situation when transferring DAQ lists.

With EV_SESSION_TERMINATED the slave indicates to the master that it autonomously decided to disconnect the current *X*CP session.

EV_USER is a carrier for user-defined events.

EV_TRANSPORT is a carrier for Transport Layer specific events.

1.3 Table of Service Request codes (SERV)

The service request packet codes in the table below may be sent as an asynchronous packet with PID 0xFC.

The implementation is optional for the slave device, but mandatory for the master device. Service request packets sent from the slave device to the master device are not acknowledged, therefore the transmission is not guaranteed.

Service Request	Code	Description
SERV_RESET	0x00	Slave requesting to be reset
SERV_TEXT	0x01	The remaining data bytes of the packet contain plain ASCII text. The line separator is LF or CR/LF. The text must be null terminated to indicate the end of the overall packet.

1.4 Table of Command codes (CMD)

An attempt to execute a not implemented optional command will return ERR_CMD_UNKNOWN and does not have any effect.
This lets the master device detect not implemented optional commands easily.

If GET_SEED is implemented, UNLOCK is required.

If SET_CAL_PAGE is implemented, GET_CAL_PAGE is required.

If SET_DAQ_ID is implemented, GET_DAQ_ID is required.

1.4.1 Standard commands (STD)

Command	Code	Is Optional
CONNECT	0xFF	no
DISCONNECT	0xFE	no
GET_STATUS	0xFD	no
SYNCH	0xFC	no

Command	Code	Is Optional
GET_COMM_MODE_INFO	0xFB	yes
GET_ID	0xFA	yes
SET_REQUEST	0xF9	yes
GET_SEED	0xF8	yes
UNLOCK	0xF7	yes
SET_MTA	0xF6	yes
UPLOAD	0xF5	yes
SHORT_UPLOAD	0xF4	yes
BUILD_CHECKSUM	0xF3	yes

Command	Code	Is Optional
TRANSPORT_LAYER_CMD	0xF2	yes
USER_CMD	0xF1	yes

1.4.2 Calibration commands (CAL)

Command	Code	Is Optional
DOWNLOAD	0xF0	no

Command	Code	Is Optional
DOWNLOAD_NEXT	0xEF	yes
DOWNLOAD_MAX	0xEE	yes
SHORT_DOWNLOAD	0xED	yes
MODIFY_BITS	0xEC	yes

1.4.3 Page switching commands (PAG)

Command	Code	Is Optional
SET_CAL_PAGE	0xEB	no
GET_CAL_PAGE	0xEA	no

Command	Code	Is Optional
GET_PAG_PROCESSOR_INFO	0xE9	yes
GET_SEGMENT_INFO	0xE8	yes
GET_PAGE_INFO	0xE7	yes
SET_SEGMENT_MODE	0xE6	yes
GET_SEGMENT_MODE	0xE5	yes
COPY_CAL_PAGE	0xE4	yes

1.4.4 Data Acquisition and Stimulation commands (DAQ)

Command	Code	Is Optional
CLEAR_DAQ_LIST	0xE3	no
SET_DAQ_PTR	0xE2	no
WRITE_DAQ	0xE1	no
SET_DAQ_LIST_MODE	0xE0	no
GET_DAQ_LIST_MODE	0xDF	no
START_STOP_DAQ_LIST	0xDE	no
START_STOP_SYNCH	0xDD	no

Command	Code	Is Optional
GET_DAQ_CLOCK	0xDC	yes
READ_DAQ	0xDB	yes
GET_DAQ_PROCESSOR_INFO	0xDA	yes
GET_DAQ_RESOLUTION_INFO	0xD9	yes
GET_DAQ_LIST_INFO	0xD8	yes
GET_DAQ_EVENT_INFO	0xD7	yes

Command	Code	Is Optional
FREE_DAQ	0xD6	yes
ALLOC_DAQ	0xD5	yes
ALLOC_ODT	0xD4	yes
ALLOC_ODT_ENTRY	0xD3	yes

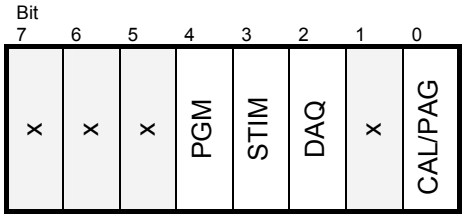
1.4.5 Non-volatile memory programming commands (PGM)

Command	Code	Is Optional
PROGRAM_START	0xD2	no
PROGRAM_CLEAR	0xD1	no
PROGRAM	0xD0	no
PROGRAM_RESET	0xCF	no

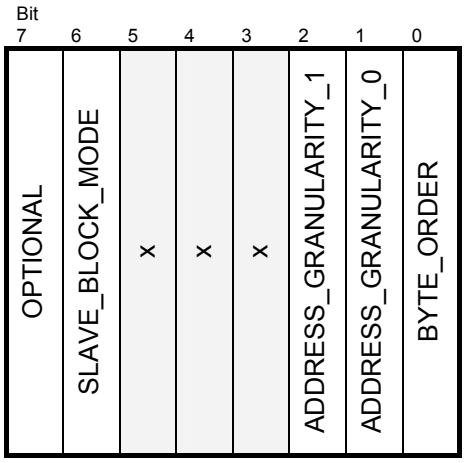
Command	Code	Is Optional
GET_PGM_PROCESSOR_INFO	0xCE	yes
GET_SECTOR_INFO	0xCD	yes
PROGRAM_PREPARE	0xCC	yes
PROGRAM_FORMAT	0xCB	yes
PROGRAM_NEXT	0xCA	yes
PROGRAM_MAX	0xC9	yes
PROGRAM_VERIFY	0xC8	yes

1.5 Table of bit mask coded parameters

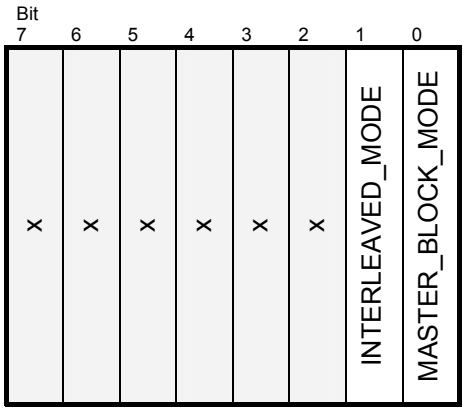
RESSOURCE parameter in CONNECT and GET_SEED



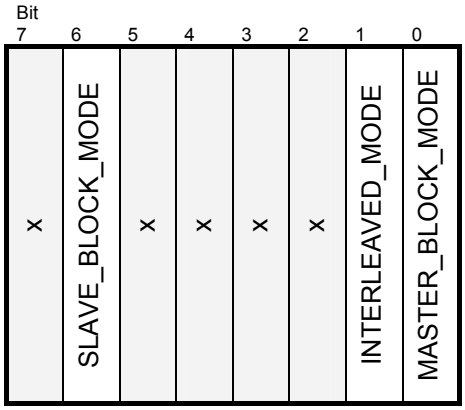
COMM_MODE_BASIC parameter in CONNECT



COMM_MODE_OPTIONAL parameter in GET_COMM_MODE_INFO



COMM_MODE_PGM parameter in PROGRAM_START

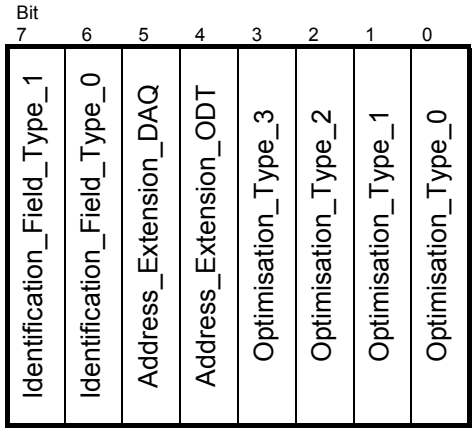


Current Resource Protection Status parameter in GET_STATUS and UNLOCK

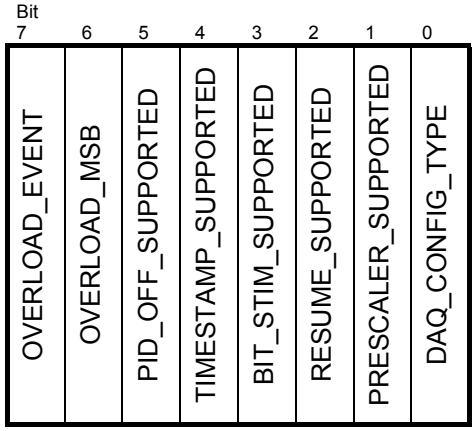
Mode parameter in SET_REQUEST

Current Session Status parameter in GET_STATUS

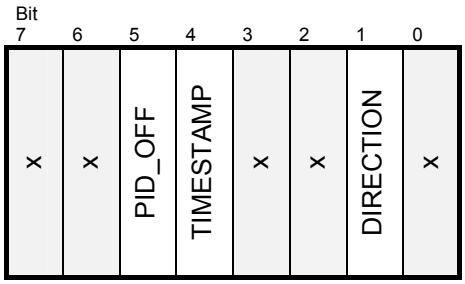
DAQ_KEY_BYTE parameter in GET_DAQ_PROCESSOR_INFO



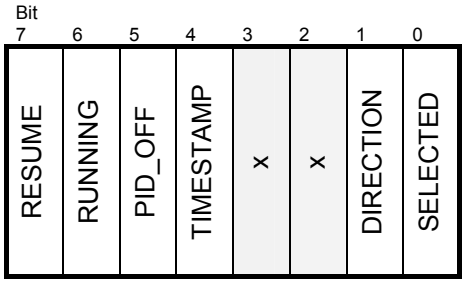
DAQ_PROPERTIES parameter in GET_DAQ_PROCESSOR_INFO



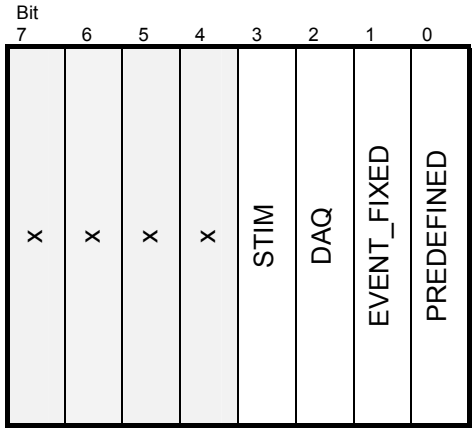
Mode parameter in SET_DAQ_LIST_MODE



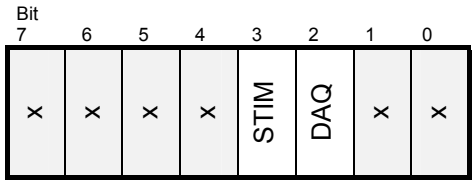
Current Mode parameter in GET_DAQ_LIST_MODE



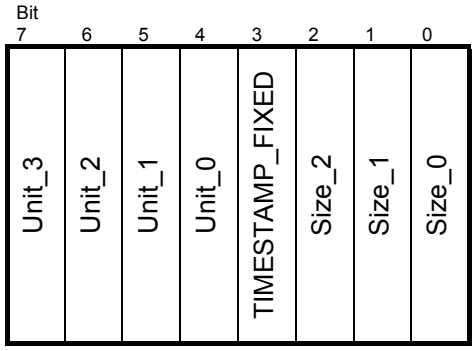
DAQ_LIST_PROPERTIES parameter in GET_DAQ_LIST_INFO



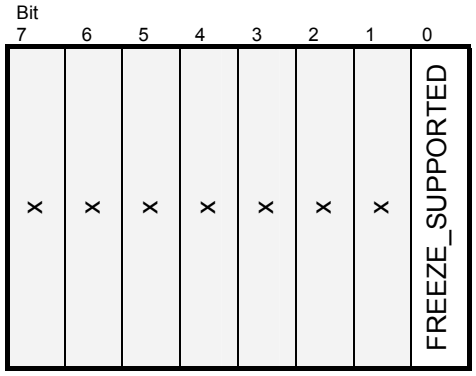
DAQ_EVENT_PROPERTIES parameter in GET_DAQ_EVENT_INFO



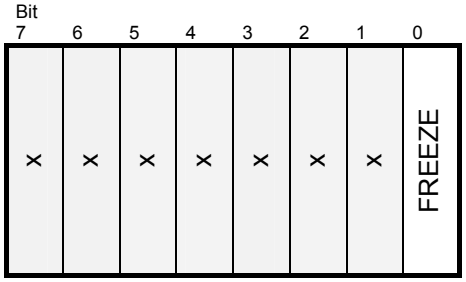
TIMESTAMP_MODE parameter in GET_DAQ_RESOLUTION_INFO



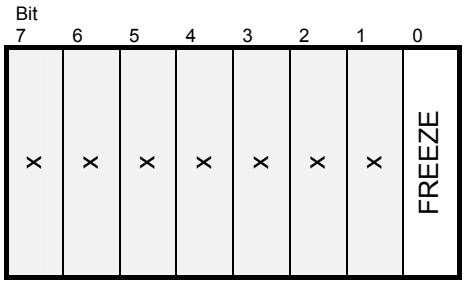
PAG_PROPERTIES parameter in GET_PAG_PROCESSOR_INFO



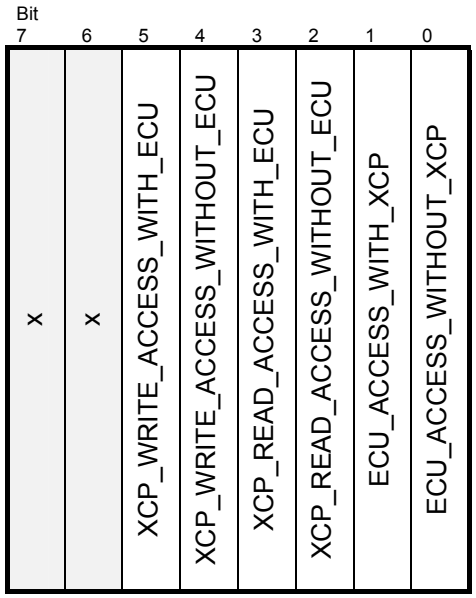
Mode parameter in SET_SEGMENT_MODE



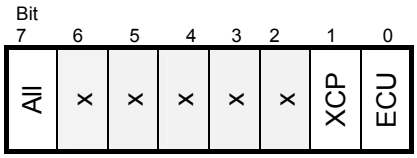
Current Mode parameter in GET_SEGMENT_MODE



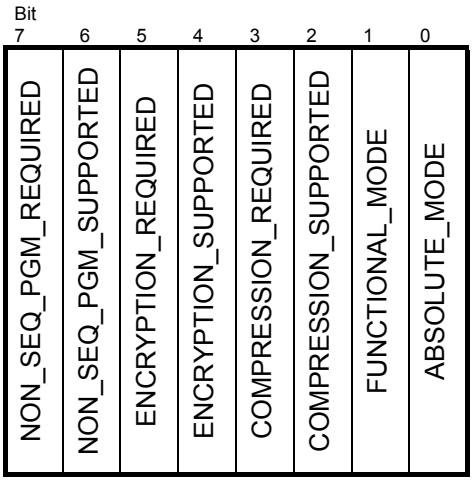
PAGE_PROPERTIES parameter in GET_PAGE_INFO



Mode parameter in SET_CAL_PAGE:



PGM_PROPERTIES parameter in GET_PGM_PROCESSOR_INFO



1.6 Description of Commands

The following chapters are a description of all possible *X*CP command packets and their responses.

Unused data bytes, marked as „reserved", may have arbitrary values.

Command parameters in WORD (2 Byte) format, are always aligned to a position that can be divided by 2. Command parameters in DWORD (4 Bytes) format, are always aligned to a position that can be divided by 4.

The byte format (MOTOROLA, INTEL) of multi byte parameters is slave device dependent.

The structure of the command description is always as follows:

Command CMD:

Position	Type	Description
0	BYTE	Command Packet Code CMD
1..MAX_CTO-1	BYTE	Command specific Parameters

Command Positive Response RES:

Position	Type	Description
0	BYTE	Command Positive Response Packet Code = RES 0xFF
1..MAX_CTO-1	BYTE	Command specific Parameters

Command Negative Response ERR:

Position	Type	Description
0	BYTE	Error Packet Code = 0xFE
1	BYTE	Error code
2..MAX_CTO-1	BYTE	Command specific Parameters

To simplify this documentation, in the following sections of this document, positive and negative responses are not explicitly described unless they have parameters.

1.6.1 Standard commands (STD)

1.6.1.1 Mandatory commands

1.6.1.1.1 Set up connection with slave

Category Standard, mandatory
Mnemonic CONNECT

Position	Type	Description
0	BYTE	Command Code = 0xFF
1	BYTE	Mode 00 = Normal 01 = user defined

This command establishes a ***continuous, logical, point-to-point connection*** with a slave device.

During a running *X*CP session (CONNECTED) this command has no influence on any configuration of the *X*CP slave driver.

A slave device does not respond to any other commands (except auto detection) unless it is in the state CONNECTED.

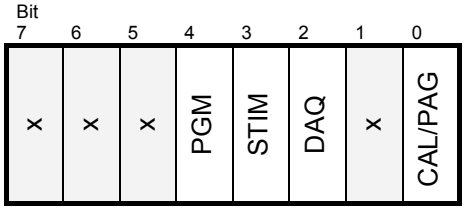
With a CONNECT(Mode = Normal), the master can start an *X*CP communication with the slave.

With a CONNECT(Mode = user defined), the master can start an *X*CP communication with the slave and at the same time tell the slave that it should go into a special (user defined) mode.

Positive Response:

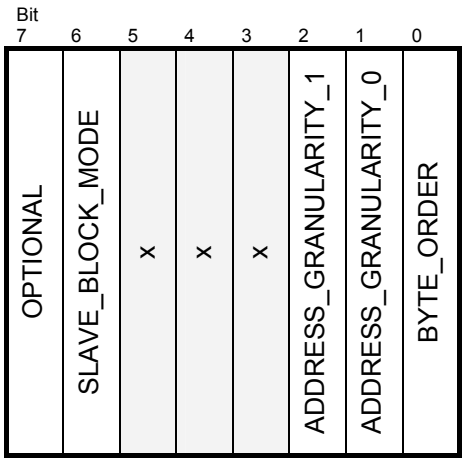
Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	RESSOURCE
2	BYTE	COMM_MODE_BASIC
3	BYTE	MAX_CTO, Maximum CTO size [BYTE]
4,5	WORD	MAX_DTO, Maximum DTO size [BYTE]
6	BYTE	XCP Protocol Layer Version Number (most significant byte only)
7	BYTE	XCP Transport Layer Version Number (most significant byte only)

The RESSOURCE parameter is a bit mask described below :



Flag	Description
CAL/PAG	CALibration and PAGing 0 = calibration/ paging not available 1 = calibration/ paging available The commands DOWNLOAD, DOWNLOAD_MAX, SHORT_DOWNLOAD, SET_CAL_PAGE, GET_CAL_PAGE are available.
DAQ	DAQ lists supported 0 = DAQ lists not available 1 = DAQ lists available The DAQ commands (GET_DAQ_PROCESSOR_INFO, GET_DAQ_LIST_INFO, ...) are available.
STIM	STIMulation 0 = stimulation not available 1 = stimulation available data stimulation mode of a DAQ list available
PGM	ProGraMming 0 = Flash programming not available 1 = Flash programming available The commands PROGRAM_CLEAR, PROGRAM, PROGRAM_MAX are available.

The COMM_MODE_BASIC parameter is a bit mask described below :



BYTE_ORDER indicates the byte order used for transferring multi-byte parameters in an *X*CP Packet. BYTE_ORDER = 0 means Intel format, BYTE_ORDER = 1 means Motorola format. Motorola format means MSB on lower address/position.

Bit 2	Bit 1	ADDRESS_ GRANULARITY	AG [BYTE]
ADDRESS_GRANULARITY_1	ADDRESS_GRANULARITY_0		
0	0	BYTE	1
0	1	WORD	2
1	0	DWORD	4
1	1	reserved	

The address granularity indicates the size of an element contained at a single address.
It is needed if the master has to do address calculation.

Granularity	BYTE	WORD	
Address n	Byte 00	Byte 00	Byte 01
Address n+1	Byte 01	Byte 02	Byte 03

The SLAVE_BLOCK_MODE flag indicates whether the Slave Block Mode is available.
The OPTIONAL flag indicates whether additional information on supported types of Communication mode is available. The master can get that additional information with GET_COMM_MODE_INFO.

MAX_CTO is the maximum CTO packet size in bytes.
MAX_DTO is the maximum DTO packet size in bytes.

The following relations must always be fulfilled

$$MAX\_CTO \bmod AG = 0$$
$$MAX\_DTO \bmod AG = 0$$

All length information which refers to the address range of the slave itself is based on the AG (ELEMENTS). If the length information refers to the data stream (*X*CP Protocol), it is based on bytes.

The *X*CP Protocol Layer Version Number indicates the major version of the Protocol Layer Specification.

The *X*CP Transport Layer Version Number indicates the major version of the Specification of the current Transport Layer.

1.6.1.1.2 Disconnect from slave

Category Standard, mandatory
Mnemonic DISCONNECT

Position	Type	Description
0	BYTE	Command Code = 0xFE

Brings the slave to the “DISCONNECTED” state.
The “DISCONNECTED” state is described in Part 1, chapter “state machine”.

Negative Response:
If DISCONNECT is currently not possible, ERR_CMD_BUSY will be returned.

1.6.1.1.3 Get current session status from slave

Category Standard, mandatory
Mnemonic GET_STATUS

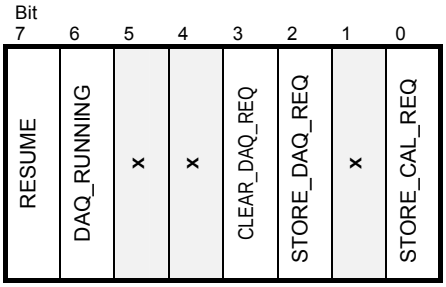
Position	Type	Description
0	BYTE	Command Code = 0xFD

This command returns all current status information of the slave device. This includes the status of the resource protection, pending store requests and the general status of data acquisition and stimulation.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID = 0xFF
1	BYTE	Current session status
2	BYTE	Current resource protection status
3	BYTE	Reserved
4,5	WORD	Session configuration id

The Current Session Status parameter is a bit mask described below:



Flag	Description
STORE_CAL_REQ	REQuest to STORE CALibration data 0 = STORE_CAL_REQ mode is reset. 1 = STORE_CAL_REQ mode is set
STORE_DAQ_REQ	REQuest to STORE DAQ list 0 = STORE_DAQ_REQ mode is reset. 1 = STORE_DAQ_REQ mode is set
CLEAR_DAQ_REQ	REQuest to CLEAR DAQ configuration 0 = CLEAR_DAQ_REQ is reset. 1 = CLEAR_DAQ_REQ is set
DAQ_RUNNING	Data Transfer 0 = Data transfer is not running 1 = Data transfer is running.
RESUME	RESUME Mode 0 = Slave is not in RESUME mode 1 = Slave is in RESUME mode

The STORE_CAL_REQ flag indicates a pending request to save the calibration data into non-volatile memory. As soon as the request has been fulfilled, the slave will reset the appropriate bit. The slave device may indicate this by transmitting an EV_STORE_CAL event packet.

The STORE_DAQ_REQ flag indicates a pending request to save the DAQ list setup into non-volatile memory. As soon as the request has been fulfilled, the slave will reset the appropriate bit. The slave device may indicate this by transmitting an EV_STORE_DAQ event packet.

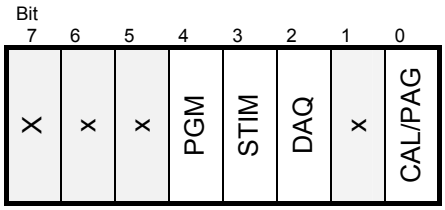
The CLEAR_DAQ_REQ flag indicates a pending request to clear all DAQ lists in non-volatile memory. All ODT entries reset to address = 0, extension = 0, size = 0 and bit_offset = FF. Session configuration ID reset to 0. As soon as the request has been fulfilled, the slave will reset the appropriate bit. The slave device may indicate this by transmitting an EV_CLEAR_DAQ event packet.

If the slave device does not support the requested mode, an ERR_OUT_OF_RANGE will be returned.

The DAQ_RUNNING flag indicates that at least one DAQ list has been started and is in RUNNING mode.

The RESUME flag indicates that the slave is in RESUME mode.

The Current Resource Protection Status parameter is a bit mask described below:



Flag	Protected Commands
CAL/PAG	CALibration/PAGing commands 0 = CALibration/PAGing commands are not protected with SEED & Key mechanism 1 = CALibration/PAGing commands are protected with SEED & Key mechanism
DAQ	DAQ list commands (DIRECTION = DAQ) 0 = DAQ list commands are not protected with SEED & Key mechanism 1 = DAQ list commands are protected with SEED & Key mechanism
STIM	DAQ list commands (DIRECTION = STIM) 0 = DAQ list commands are not protected with SEED & Key mechanism 1 = DAQ list commands are protected with SEED & Key mechanism
PGM	ProGraMming commands 0 = ProGraMming commands are not protected with SEED & Key mechanism 1 = ProGraMming commands are protected with SEED & Key mechanism

The commands of the STanDard group are NEVER protected

Command
CONNECT
DISCONNECT
GET_STATUS
SYNCH

Command
GET_COMM_MODE_INFO
GET_ID
SET_REQUEST
GET_SEED
UNLOCK
SET_MTA
UPLOAD
SHORT_UPLOAD
BUILD_CHECKSUM

Command
TRANSPORT_LAYER_CMD
USER_CMD

The CAL/PAG flags indicates that all commands of the CALibration/PAGing group are protected and will return an ERR_ACCESS_LOCKED upon an attempt to execute the command without a previous successful GET_SEED/UNLOCK sequence.

Command
DOWNLOAD

Command
DOWNLOAD_NEXT
DOWNLOAD_MAX
SHORT_DOWNLOAD
MODIFY_BITS

Command
SET_CAL_PAGE
GET_CAL_PAGE

Command
GET_PAG_PROCESSOR_INFO
GET_SEGMENT_INFO
GET_PAGE_INFO
SET_SEGMENT_MODE
GET_SEGMENT_MODE
COPY_CAL_PAGE

The DAQ flags indicates that the following commands of the Data AcQuisition and stimulation group are protected and will return a ERR_ACCESS_LOCKED upon an attempt to execute the command without a previous successful GET_SEED/UNLOCK sequence.

Command
CLEAR_DAQ_LIST
SET_DAQ_PTR
WRITE_DAQ
SET_DAQ_LIST_MODE
GET_DAQ_LIST_MODE
START_STOP_DAQ_LIST
START_STOP_SYNCH

Command
GET_DAQ_CLOCK
READ_DAQ
GET_DAQ_PROCESSOR_INFO
GET_DAQ_RESOLUTION_INFO
GET_DAQ_LIST_INFO
GET_DAQ_EVENT_INFO

Command
FREE_DAQ
ALLOC_DAQ
ALLOC_ODT
ALLOC_ODT_ENTRY

The PGM flags indicates that all the commands of the ProGraMming group are protected and will return a ERR_ACCESS_LOCKED upon an attempt to execute the command without a previous successful GET_SEED/UNLOCK sequence.

Command
PROGRAM_START
PROGRAM_CLEAR
PROGRAM
PROGRAM_RESET

Command
GET_PGM_PROCESSOR_INFO
GET_SECTOR_INFO
PROGRAM_PREPARE
PROGRAM_FORMAT
PROGRAM_NEXT
PROGRAM_MAX
PROGRAM_VERIFY

Session configuration id:

The session configuration id has to be set by a prior SET_REQUEST command with STORE_DAQ_REQ set. This allows the master device to verify that automatically started DAQ lists contain the expected data transfer configuration.

1.6.1.1.4 Synchronize command execution after time-out

Category Standard, mandatory
Mnemonic SYNCH

Position	Type	Description
0	BYTE	Command Code = 0xFC

This command is used to synchronize command execution after timeout conditions. The SYNCH command will always have a negative response with the error code ERR_CMD_SYNCH. There is no other command using this error code, therefore the response to a SYNCH command may be distinguished from the response to any other command.

For a detailed explanation of the purpose of the SYNCH command, please refer to the chapter “time-out handling”.

Negative Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFE
1	BYTE	Error Code = ERR_CMD_SYNCH

1.6.1.2 Optional commands

1.6.1.2.1 Get communication mode info

Category Standard, optional
Mnemonic GET_COMM_MODE_INFO

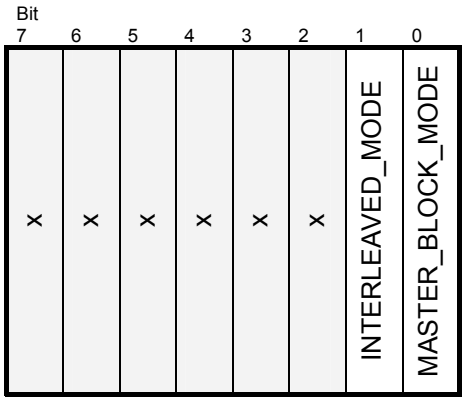
Position	Type	Description
0	BYTE	Command Code = 0xFB

This command returns optional information on different Communication Modes supported by the slave.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Reserved
2	BYTE	COMM_MODE_OPTIONAL
3	BYTE	Reserved
4	BYTE	MAX_BS
5	BYTE	MIN_ST
6	BYTE	QUEUE_SIZE
7	BYTE	XCP Driver Version Number

The COMM_MODE_OPTIONAL parameter is a bit mask described below :



The MASTER_BLOCK_MODE flag indicates whether the Master Block Mode is available.

If the master device block mode is supported, MAX_BS indicates the maximum allowed block size as the number of consecutive command packets (DOWNLOAD_NEXT or PROGRAM_NEXT) in a block sequence. MIN_ST indicates the required minimum separation time between the packets of a block transfer from the master device to the slave device in units of 100 microseconds.

The INTERLEAVED_MODE flag indicates whether the Interleaved Mode is available.

If interleaved mode is available, QUEUE_SIZE indicates the maximum number of consecutive command packets the master can send to the receipt queue of the slave.

The *X*CP Driver Version Number indicates the version number of the *X*CP driver in the slave.

The major driver version is the high nibble of the version number, the minor driver version is the low nibble.

1.6.1.2.2 Get identification from slave

Category Standard, optional
Mnemonic GET_ID

Position	Type	Description
0	BYTE	Command Code = 0xFA
1	BYTE	Requested Identification Type

This command is used for automatic session configuration and for slave device identification.
The following identification types may be requested:

Type	Description
0	ASCII text
1	ASAM-MC2 filename without path and extension
2	ASAM-MC2 filename with path and extension
3	URL where the ASAM-MC2 file can be found
4	ASAM-MC2 file to upload
128..255	User defined

Which types are supported by the slave device is implementation specific.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Mode
2,3	WORD	Reserved
4..7	DWORD	Length [BYTE]
7+...	ELEMENT	Identification (if mode = 1)

The parameter Length specifies the number of bytes in the identification. If length is 0, the requested identification type is not available.

ELEMENT is BYTE, WORD or DWORD, depending on the AG.

If mode is 1, the identification is transferred in the remaining bytes of the response.

If mode is 0, the slave device sets the *Memory Transfer Address* (MTA) to the location from which the master device may upload the requested identification using one or more UPLOAD commands.

The identification string is ASCII text format, it does not have 0 termination.

Examples:

1: test

2: c:\database\test.a2l

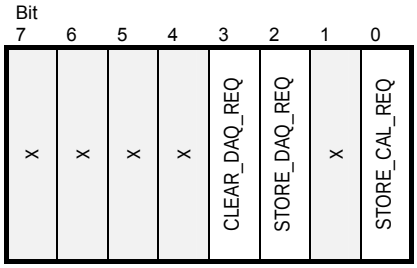
3: ftp://ttp.oem.com\data_repository\project_xcp\test.a2l

1.6.1.2.3 Request to save to non-volatile memory

Category Standard, optional
Mnemonic SET_REQUEST

Position	Type	Description
0	BYTE	Command Code = 0xF9
1	BYTE	Mode
2,3	WORD	Session configuration id

The Mode parameter is a bit mask described below:



Flag	Description
STORE_CAL_REQ	REQuest to STORE CALibration data 0 = STORE_CAL_REQ is not set. 1 = STORE_CAL_REQ is set
STORE_DAQ_REQ	REQuest to STORE DAQ list 0 = STORE_DAQ_REQ is not set. 1 = STORE_DAQ_REQ is set
CLEAR_DAQ_REQ	REQuest to CLEAR DAQ configuration 0 = CLEAR_DAQ_REQ is not set. 1 = CLEAR_DAQ_REQ is set

STORE_CAL_REQ sets a request to save calibration data into non-volatile memory. The STORE_CAL_REQ bit obtained by GET_STATUS will be reset by the slave, when the request is fulfilled. The slave device may indicate this by transmitting an EV_STORE_CAL event packet.

STORE_DAQ_REQ sets a request to save all DAQ lists, which have been selected with START_STOP_DAQ_LIST(Select) into non-volatile memory. The slave also has to store the session configuration id in non-volatile memory.

Upon saving, the slave first has to clear any DAQ list configuration that might already be stored in non-volatile memory.

The STORE_DAQ_REQ bit obtained by GET_STATUS will be reset by the slave, when the request is fulfilled. The slave device may indicate this by transmitting an EV_STORE_DAQ event packet.

CLEAR_DAQ_REQ is used to clear all DAQ lists in non-volatile memory. All ODT entries reset to address = 0, extension = 0, size = 0 and bit_offset = FF. Session configuration ID reset to 0.

The CLEAR_DAQ_REQ bit obtained by GET_STATUS will be reset by the slave, when the request is fulfilled. The slave device may indicate this by transmitting an EV_CLEAR_DAQ event packet.

If the slave device does not support the requested mode, an ERR_OUT_OF_RANGE will be returned.

1.6.1.2.4 Get seed for unlocking a protected resource

Category Standard, optional (ref. UNLOCK)
Mnemonic GET_SEED

Position	Type	Description
0	BYTE	Command Code = 0xF8
1	BYTE	Mode 0 = (first part of) seed 1 = remaining part of seed
2	BYTE	Mode=0: Resource Mode=1: Don't care

With Mode = 0, the master requests the slave to transmit (the first part of) the seed. The slave answers with (the first part of) the seed and the total length of the seed.

With Mode = 1, the master has to request the remaining part(s) of the seed from the slave if the total length of the seed is bigger than MAX_CTO-2.

The master has to use GET_SEED(Mode=1) in a defined sequence together with GET_SEED(Mode=0). If the master sends a GET_SEED(Mode=1) directly without a previous GET_SEED(Mode=0), the slave returns an ERR_SEQUENCE as negative response.

See command GET_STATUS (resource protection status) for a description for the values of the resource parameter (CAL/PAG, DAQ, STIM, PGM) and the related commands.

Only one resource may be requested with one GET_SEED command. If more than one resource has to be unlocked, the (GET_SEED+UNLOCK) sequence has to be performed multiple times. If the master does not request any resource or requests multiple resources at the same time, the slave will respond with an ERR_OUT_OF_RANGE.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Length of seed [BYTE] Length = 0 resource unprotected Mode = 0 : total length of seed Mode = 1 : remaining length of seed
2..MAX_CTO-1	BYTE	Seed

Length indicates the (remaining) number of seed bytes. If Length = 0, the resource is unprotected and no UNLOCK command is necessary.

A GET_SEED sequence returns the ´seed´ data for a ***Seed&Key*** algorithm computing the ´key´ to unlock the requested resource category for authorized access (see the UNLOCK command).

The master has to calculate the key by calling an external function file. There's only 1 external function file which might contain from 1 up to 4 different algorithms, one algorithm for each of the resources CAL/PAG, DAQ, STIM or PGM.

The external function file supplier can enable/disable the use of each of these 4 algorithms. The master can get the information about the ability of the algorithms directly from the external function file.

The external function file supplier can compile different versions of the external function file by making different combinations of enabled algorithms.

The master gets the name of the external function file to be used for this slave, from the ASAM MCD 2MC description file. The API for communicating with the external function file is specified in Part 4 "Interfacing" of the *X*CP specification.

1.6.1.2.5 Send key for unlocking a protected resource

Category Standard, optional (ref. GET_SEED)
Mnemonic UNLOCK

Position	Type	Description
0	BYTE	Command Code = 0xF7
1	BYTE	(remaining) Length of key in bytes
2..MAX_CTO-1	BYTE	Key

Unlocks the slave device's security protection using a ´key´ computed from the ´seed´ obtained by a previous GET_SEED sequence. See the description of the GET_SEED command.

Length indicates the (remaining) number of key bytes.

The master has to use UNLOCK in a defined sequence together with GET_SEED.

The master only can send an UNLOCK sequence if previously there was a GET_SEED sequence.

The master has to send the first UNLOCK after a GET_SEED sequence with a Length containing the total length of the key.

If the total length of the key is bigger than MAX_CTO-2, the master has to send the remaining key bytes with (a) consecutive UNLOCK command(s) containing the remaining length of the key.

If the master does not respect this sequence, the slave returns an ERR_SEQUENCE as negative response.

The key is checked after completion of the UNLOCK sequence. If the key is not accepted, ERR_ACCESS_LOCKED will be returned. The slave device will then go to disconnected state. A repetition of an UNLOCK sequence with a correct key will have a positive response and no other effect.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Current resource protection status

The answer upon UNLOCK contains the Current Resource Protection Mask as described at GET_STATUS.

Example 1 :

MAX_CTO	= 8 bytes (CAN)
TotalLengthOf(seed)	= 4 bytes
TotalLengthOf(key)	= 2 bytes
Seed	= 11 22 33 44
Key	= 43 21

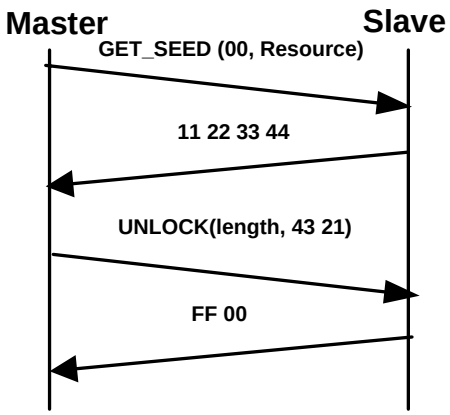


Diagram 17 : short GET_SEED+UNLOCK sequence

Example 2 :

MAX_CTO = 8 bytes (CAN)
TotalLengthOf(seed) = 19 bytes
TotalLengthOf(key) = 10 bytes

Seed = 99 88 77 66 55 44 33 22 11 00 11 22 33 44 55 66 77 88 99
Key = 98 76 54 32 10 01 23 45 67 89

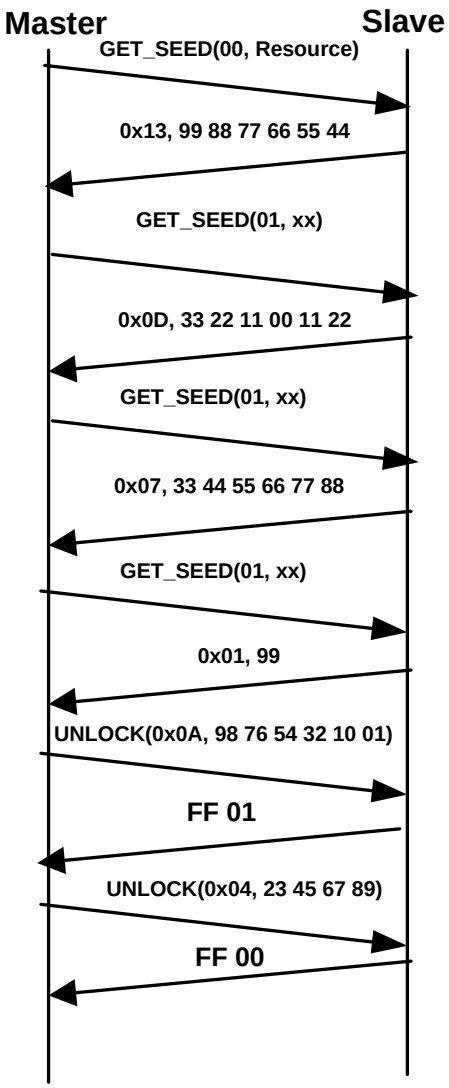


Diagram 18 : long GET_SEED+UNLOCK sequence

1.6.1.2.6 Set Memory Transfer Address in slave

Category Standard, optional
Mnemonic SET_MTA

Position	Type	Description
0	BYTE	Command Code = 0xF6
1,2	BYTE	Reserved
3	BYTE	Address extension
4.7	DWORD	Address

This command will initialize a pointer (32Bit address + 8Bit extension) for following memory transfer commands.

The MTA is used by the commands BUILD_CHECKSUM, UPLOAD, DOWNLOAD, DOWNLOAD_MAX, MODIFY_BITS, PROGRAM_CLEAR, PROGRAM and PROGRAM_MAX.

1.6.1.2.7 Upload from slave to master

Category Standard, optional
Mnemonic UPLOAD

Position	Type	Description
0	BYTE	Command Code = 0xF5
1	BYTE	Number of data elements [AG] [1..MAX_CTO/AG -1] Standard mode [1..255] Block mode

A data block of the specified length, starting at the current MTA, will be returned. The MTA will be post-incremented by the given number of data elements.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1..AG-1	BYTE	Used for alignment only if AG > 1
AG..MAX_CTO-AG	ELEMENT	Data elements

Depending on AG 1, 2 or 3 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE. WORD or DWORD, depending upon AG.

If the slave device does not support block transfer mode, all uploaded data are transferred in a single response packet. Therefore the number of data elements parameter in the request has to be in the range [1..MAX_CTO-1]. An ERR_OUT_OF_RANGE will be returned, if the number of data elements is more than MAX_CTO-1.

If block transfer mode is supported, the uploaded data are transferred in multiple responses on the same request packet. For the master there are no limitations allowed concerning the maximum block size. Therefore the number of data elements (n) can be in the range [1..255]. The slave device will transmit (n*AG / (MAX_CTO-1)) +1 response packets. The separation time between the response packets is depending on the slave device implementation. It's the responsibility of the master device to keep track of all packets and to check for lost packets. It is slave device implementation specific if the data in different response packets are consistent. For instance, this has to be considered, when block upload mode is used to obtain 8 byte floating point objects.

Examples:

MAX_CTO=8
AG=1

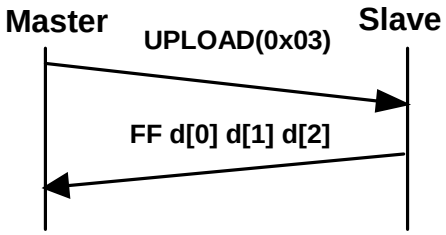


Diagram 19 : UPLOAD 3 bytes

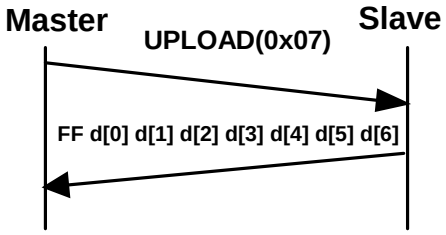


Diagram 20 : UPLOAD 7 bytes

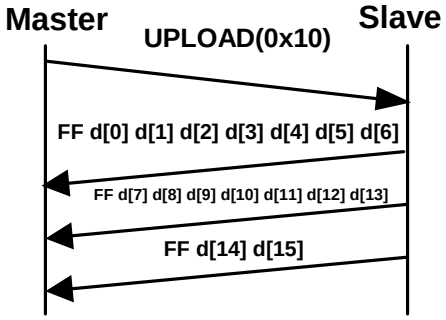


Diagram 21 : UPLOAD 16 bytes in block mode

1.6.1.2.8 Upload from slave to master (short version)

Category Standard, optional
Mnemonic SHORT_UPLOAD

Position	Type	Description
0	BYTE	Command Code = 0xF4
1	BYTE	Number of data elements [AG] [1..MAX_CTO-1]
2	BYTE	Reserved
3	BYTE	Address extension
4..7	DWORD	Address

A data block of the specified length, starting at address will be returned. The MTA pointer is set to the first data byte behind the uploaded data block. The error handling and the response structure is identical to the UPLOAD command.

ELEMENT is BYTE. WORD or DWORD, depending upon AG.

This command does not support block transfer and it must not be used within a block transfer sequence.

1.6.1.2.9 Build checksum over memory range

Category Standard, optional
Mnemonic BUILD_CHECKSUM

Position	Type	Description
0	BYTE	Command Code = 0xF3
1..3	BYTE	reserved
4..7	DWORD	Block size [AG]

Returns a checksum result of the memory block that is defined by the MTA and Block size. The MTA will be post-incremented by the block size.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Checksum type
2,3	BYTE	Reserved
4..7	DWORD	Checksum

The following checksum types are defined:

Type	Name	Description
0x01	XCP_ADD_11	Add BYTE into a BYTE checksum, ignore overflows
0x02	XCP_ADD_12	Add BYTE into a WORD checksum, ignore overflows
0x03	XCP_ADD_14	Add BYTE into a DWORD checksum, ignore overflows
0x04	XCP_ADD_22	Add WORD into a WORD checksum, ignore overflows, blocksize must be modulo 2
0x05	XCP_ADD_24	Add WORD into a DWORD checksum, ignore overflows, blocksize must be modulo 2
0x06	XCP_ADD_44	Add DWORD into DWORD, ignore overflows, blocksize must be modulo 4

Type	Name	Description
0x07	XCP_CRC_16	See CRC error detection algorithms
0x08	XCP_CRC_16_CITT	See CRC error detection algorithms
0x09	XCP_CRC_32	See CRC error detection algorithms

Type	Name	Description
0xFF	XCP_USER_DEFINED	User defined algorithm, in externally calculated function

The result is always given as a DWORD, regardless of the checksum type.

With the Checksum Type “XCP_USER_DEFINED”, the Slave can indicate that the Master for calculating the checksum has to use a user-defined algorithm implemented in an externally calculated function (e.g.Win32 DLL, UNIX ® shared object file,..)

The master gets the name of the external function file to be used for this slave, from the ASAM MCD 2MC description file.

The API for communicating with the external function file is specified in Part 4 “Interfacing” of the *X*CP specification.

Negative Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFE
1	BYTE	Error code
2,3	WORD	reserved
4..7	DWORD	Maximum block size [AG]

If the blocksize exceeds the allowed maximum value, an ERR_OUT_OF_RANGE will be returned. The maximum block size will be returned in the checksum field.

The CRC algorithms are specified by the following parameters :

Name	Width	Poly	Init	Refin	Refout	XORout
XCP_CRC_16	16	0x8005	0x0000	TRUE	TRUE	0x0000
XCP_CRC16_CITT	16	0x1021	0xFFFF	FALSE	FALSE	0x0000
XCP_CRC_32	32	0x04C11DB7	0xFFFFFFFF	TRUE	TRUE	0xFFFFFFFF

Name:
this is the name given to the algorithm. A string value starting with “XCP_”.

Width:
this is the width of the algorithm expressed in bits. This is one less than the width of the poly.

Poly:
This parameter is the polynomial. This is a binary value that should be specified as a hexadecimal number. The top bit of the poly should be omitted. For example, if the poly is 10110, you should specify 0x06. An important aspect of this parameter is that it represents the unreflected poly; the bottom of this parameter is always the LSB of the divisor during the division, regardless of whether the algorithm is reflected.

Init:
This parameter specifies the initial value of the register when the algorithm starts. This is the value that is to be assigned to the register in the direct table algorithm. In the table algorithm, we may think of the register always commencing with the value zero, and this value being XORed into the register after the N’th bit iteration. This parameter should be specified as a hexadecimal number.

Refin:
This is a Boolean parameter. If it is FALSE, input bytes are processed with bit 7 being treated as the most significant bit (MSB) and bit 0 being treated as the least significant bit. If this parameter is TRUE, each byte is reflected before being processed.

Refout:
This is a boolean parameter. If it is set to FALSE, the final value in the register is fed into the XORout stage directly. If this parameter is TRUE, the final register value is reflected first.

XORout:
This is a width-bit value that should be specified as hexadecimal number. It is XORed to the final register value (after the Refout stage) before the value is returned as the official checksum.

For more detailed information about CRC algorithms, please refer to :

http://www.repairfaq.org/filipg/LINK/F_crc_v34.html

1.6.1.3 Auxiliary commands

1.6.1.3.1 Refer to transport layer specific command

Category Standard, auxiliary
Mnemonic TRANSPORT_LAYER_CMD

Position	Type	Description
0	BYTE	Command Code = 0xF2
1	BYTE	Sub command code
2...	BYTE	Parameters

This command is defined in the Transport Layer specification. It is used to perform Transport Layer specific actions.

Example:

Category CAN only, optional
Mnemonic GET_SLAVE_ID

Position	Type	Description
0	BYTE	Command Code = TRANSPORT_LAYER_CMD = 0xF2
1	BYTE	Sub Command Code = 0xFF
2	BYTE	0x58 (ASCII = X)
3	BYTE	0x43 (ASCII = C)
4	BYTE	0x50 (ASCII = P)
5	BYTE	Mode 0 = identify by echo 1 = confirm by inverse echo

1.6.1.3.2 Refer to user defined command

Category Standard, auxiliary
Mnemonic USER_CMD

Position	Type	Description
0	BYTE	Command Code = 0xF1
1	BYTE	Sub command code
2...	BYTE	Parameters

This command is user defined. It mustn't be used to implement functionalities done by other services.

1.6.2 Calibration commands (CAL)

1.6.2.1 Mandatory commands

1.6.2.1.1 Download from master to slave

Category Calibration, mandatory
Mnemonic DOWNLOAD

Position	Type	Description
0	BYTE	Command Code = 0xF0
1	BYTE	Number of data elements [AG] [1..(MAX_CTO-2)/AG] Standard mode [1..min(MAX_BS*(MAX_CTO-2)/AG,255)] Block mode
2..AG-1	BYTE	Used for alignment, only if AG >2
AG=1: 2..MAX_CTO-2 AG>1: AG MAX_CTO-AG	ELEMENT	Data elements

If AG = DWORD, 2 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE, WORD or DWORD depending upon AG.

The data block of the specified length (size) contained in the CMD will be copied into memory, starting at the MTA. The MTA will be post-incremented by the number of data bytes.

If the slave device does not support block transfer mode, all downloaded data are transferred in a single command packet. Therefore the number of data elements parameter in the request has to be in the range [1..MAX_CTO-2]. An ERR_OUT_OF_RANGE will be returned, if the number of data elements is more than MAX_CTO-2.

If block transfer mode is supported, the downloaded data are transferred in multiple command packets. For the slave however, there might be limitations concerning the maximum number of consecutive command packets (block size MAX_BS). Therefore the number of data elements (n) can be in the range [1..min(MAX_BS*(MAX_CTO-2)/AG,255)].

The master device has to transmit (n * AG / (MAX_CTO-2)) - 1 additional, consecutive DOWNLOAD_NEXT command packets. The slave device will acknowledge only the last DOWNLOAD_NEXT command packet. The separation time between the command packets and the maximum number of packets are specified in the response for the CONNECT command (MAX_BS, MIN_ST).

Example:

MAX_CTO=8

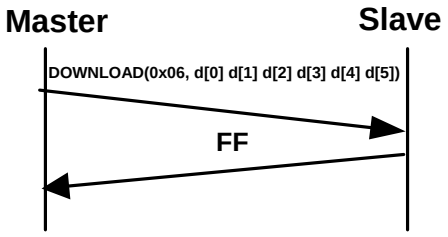


Diagram 22 : DOWNLOAD 6 bytes

1.6.2.2 Optional commands

1.6.2.2.1 Download from master to slave (Block Mode)

Category Calibration, optional
Mnemonic DOWNLOAD_NEXT

Position	Type	Description
0	BYTE	Command Code = 0xEF
1	BYTE	Number of data elements [AG] [1..min(MAX_BS*(MAX_CTO-2)/AG,255)–(MAX_CTO-2)/AG]
2..AG-1	BYTE	Used for alignment, only if AG >2
AG=1: 2..MAX_CTO-2 AG>1: AG MAX_CTO-AG	ELEMENT	Data elements

If AG = 4, 2 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE, WORD or DWORD, depending upon AG.

This command is used to transmit consecutive data elements for the DOWNLOAD command in block transfer mode.

The DOWNLOAD_NEXT command has exactly the same structure as the DOWNLOAD command. It contains the remaining number of data elements to transmit. The slave device will use this information to detect lost packets. If a sequence error has been detected, the error code ERR_SEQUENCE will be returned.

Negative Response:

If the number of data elements does not match the expected value, the error code ERR_SEQUENCE will be returned. The negative response will contain the expected number of data elements.

Position	Type	Description
0	BYTE	Packet ID: 0xFE
1	BYTE	ERR_SEQUENCE
2	BYTE	Number of expected data elements

Example:

MAX_CTO=8

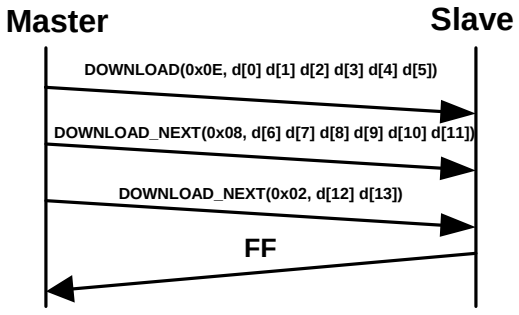


Diagram 23 : DOWNLOAD 14 bytes in block mode

1.6.2.2.2 Download from master to slave (fixed size)

Category Calibration, optional
Mnemonic DOWNLOAD_MAX

Position	Type	Description
0	BYTE	Command Code = 0xEE
1..AG-1	BYTE	Used for alignment, only if AG >1
AG..MAX_CTO-AG	ELEMENT	Data elements

Depending upon AG, 1 or 3 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE, WORD or DWORD, depending upon AG.

The data block with the fixed length (size) of MAX_CTO/AG-1 elements contained in the CMD will be copied into memory, starting at the MTA. The MTA will be post-incremented by MAX_CTO/AG-1.

This command does not support block transfer and it mustn't not be used within a block transfer sequence.

1.6.2.2.3 Download from master to slave (short version)

Category Calibration, optional
Mnemonic SHORT_DOWNLOAD

Position	Type	Description
0	BYTE	Command Code = 0xED
1	BYTE	Number of data elements [0..(MAX_CTO-8)/AG]
2	BYTE	Reserved
3	BYTE	Address extension
4..7	DWORD	Address
8..MAX_CTO-1	ELEMENT	Data elements

ELEMENT is BYTE, WORD or DWORD, depending upon AG.

A data block of the specified length, starting at address will be written. The MTA pointer is set to the first data element behind the downloaded data block. If the number of elements exceeds (MAX_CTO-8)/AG, the error code ERR_OUT_OF_RANGE will be returned.

This command does not support block transfer and it mustn't be used within a block transfer sequence.

Please note that this command will have no effect (no data bytes can be transferred) if MAX_CTO = 8 (e.g. *X*CP on CAN).

1.6.2.2.4 Modify bits

Category Calibration, optional
Mnemonic MODIFY_BITS

Position	Type	Description
0	BYTE	Command Code = 0xEC
1	BYTE	Shift Value (S)
2,3	WORD	AND Mask (MA)
4,5	WORD	XOR Mask (MX)

The 32 Bit memory location A referred by the MTA will be modified using the formula below:

A = ((A) & ((~((dword)(((word)~MA)<<S))))^((dword)(MX<<S)))

The AND Mask (MA) specifies all the bits of A which have to be set to “0” by setting the corresponding bit in MA to “0” and all untouched bits to “1”.

The XOR Mask (MX) specifies all bits of A which has to be toggled by setting the corresponding bit in MX to “1” and all untouched bits to “0”.

To set bit 0 to “0”, use MA = 0xFFFE and MX = 0x0000.
To set bit 0 to “1” first set it to “0” and then toggle it, so MA = 0xFFFE and MX = 0x0001.

Via the masks MA and MX it is only possible to access a 16 bit wide memory location. Thus the shift parameter S is used to move both masks together with the specified number of bits into the more significant direction.

Example:

	MSB							LSB
A =	1111	1111	1111	0000	1111	1111	1111	1111

To set bit 30 to “0” and bit 16 to “1” the parameters are:

S=16
A = 1**1**11 1111 1111 000**0** 1111 1111 1111 1111
MA = **1011 1111 1111 1110**
MX = **0000 0000 0000 0001**

Result:
A = 1**0**11 1111 1111 000**1** 1111 1111 1111 1111

The MTA will not be affected.

1.6.3 Page switching commands (PAG)

1.6.3.1 Mandatory commands

1.6.3.1.1 Set calibration page

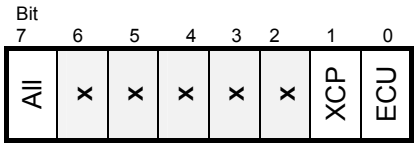
Category	Page switching, mandatory
Mnemonic	SET_CAL_PAGE

This command sets the access mode for a calibration data segment, if the slave device supports calibration data page switching (PAG flag in the resource availability mask).

Position	Type	Description
0	BYTE	Command Code = 0xEB
1	BYTE	Mode
2	BYTE	Logical data segment number
3	BYTE	Logical data page number

A calibration data segment and its pages are specified by logical numbers.

The Mode parameter is a bit mask described below:



Flag	Description
ECU	The given page will be used by the slave device application.
XCP	The slave device *X*CP driver will access the given page.
ALL	The logical segment number is ignored. The command applies to all segments

Both flags ECU and *X*CP may be set simultaneously or separately.

If the calibration data page cannot be set to the given mode, an ERR_MODE_NOT_VALID will be returned.

If the calibration data page is not available, a ERR_PAGE_NOT_VALID or ERR_SEGMENT_NOT_VALID will be returned.

1.6.3.1.2 Get calibration page

Category Page switching, mandatory
Mnemonic GET_CAL_PAGE

Position	Type	Description
0	BYTE	Command Code = 0xEA
1	BYTE	Access Mode
2	BYTE	Logical data segment number

This command returns the logical number for the calibration data page that is currently activated for the specified access mode and data segment. Mode may be 0x01 (ECU access) or 0x02 (*X*CP access). All other values are invalid.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	reserved
2	BYTE	reserved
3	BYTE	Logical data page number

1.6.3.2 Optional commands

1.6.3.2.1 Get general information on PAG processor

Category Paging, optional
Mnemonic GET_PAG_PROCESSOR_INFO

Position	Type	Description
0	BYTE	Command Code = 0xE9

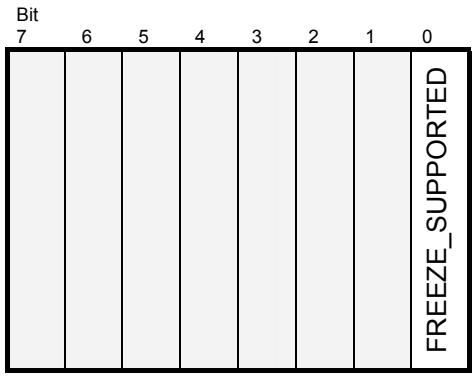
This command returns general information on paging.

Positive response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	MAX_SEGMENT total number of available segments
2	BYTE	PAG_PROPERTIES General properties for paging

MAX_SEGMENT is the total number of segments in the slave device

The PAG_PROPERTIES parameter is a bit mask described below:



Flag	Description
FREEZE_SUPPORTED	0 = SEGMENTS can not be set to FREEZE mode. 1 = SEGMENTS can be set to FREEZE mode.

The FREEZE_SUPPORTED flag indicates that all SEGMENTS can be put in FREEZE mode.

1.6.3.2.2 Get specific information for a SEGMENT

Category Page switching, optional
Mnemonic GET_SEGMENT_INFO

Position	Type	Description
0	BYTE	Command Code = 0xE8
1	BYTE	Mode 0 = get basic address info for this SEGMENT 1 = get standard info for this SEGMENT 2 = get address mapping info for this SEGMENT
2	BYTE	SEGMENT_NUMBER [0,1,..MAX_SEGMENT-1]
3	BYTE	SEGMENT_INFO Mode 0: 0 = address 1 = length Mode 1: don't care Mode 2: 0 = source address 1 = destination address 2 = length address
4	BYTE	MAPPING_INDEX [0,1,..MAX_MAPPING-1] Mode 0: don't care Mode 1: don't care Mode 2: identifier for address mapping range that MAPPING_INFO belongs to

GET_SEGMENT_INFO returns information on a specific SEGMENT.
If the specified SEGMENT is not available, ERR_OUT_OF_RANGE will be returned.

For Mode = 0 and Mode = 2, SEGMENT_INFO contains address range information.
If Mode = 1, SEGMENT_INFO is "don't care".
For Mode = 2, MAPPING_INDEX indicates the range MAPPING_INFO belongs to.
For Mode = 0 and Mode = 1, MAPPING_INDEX is "don't care"

If Mode = 0, SEGMENT_INFO indicates the kind of segment information that is requested from the slave for this SEGMENT.

Positive response: (mode = 0)

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1..3	BYTE	reserved
4..7	DWORD	BASIC_INFO 0 = address of this SEGMENT 1 = length of this SEGMENT

If Mode = 0, the response contains address information about this SEGMENT.

If SEGMENT_INFO = 0 , this command returns the address of this SEGMENT in BASIC_INFO.

If SEGMENT_INFO = 1 , this command returns the length of this SEGMENT in BASIC_INFO.

Positive response: (mode = 1)

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	MAX_PAGES number of PAGEs for this SEGMENT
2	BYTE	ADDRESS_EXTENSION address extension for this SEGMENT
3	BYTE	MAX_MAPPING number of mapped address ranges within this SEGMENT
4	BYTE	Compression method
5	BYTE	Encryption method

If Mode = 1, the response contains standard information about this SEGMENT.

MAX_PAGES indicates the number of available PAGEs for this SEGMENT.

ADDRESS_EXTENSION is used in SET_MTA, SHORT_UPLOAD and SHORT_DOWNLOAD when accessing a PAGE within this SEGMENT.

MAX_MAPPING indicates the number of address ranges within this SEGMENT that should have an address mapping applied.

The compression and the encryption method of the slave segment must correspond to the compression and the encryption method of the segment of the new flashware.

If Mode = 2, SEGMENT_INFO indicates the kind of mapping information that is requested from the slave for the range referenced by MAPPING_INDEX.

Positive response: (mode = 2)

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1..3	BYTE	reserved
4..7	DWORD	MAPPING_INFO 0 = source address for this MAPPING_INDEX 1 = destination address for this MAPPING_INDEX 2 = length for this MAPPING_INDEX

If Mode = 2, the response contains mapping information about this SEGMENT for the range indicated with MAPPING_INDEX.

If SEGMENT_INFO = 0 , this command returns the source address for this MAPPING_INDEX in MAPPING_INFO.

If SEGMENT_INFO = 1 , this command returns the destination address for this MAPPING_INDEX in MAPPING_INFO.

If SEGMENT_INFO = 2 , this command returns the length for this MAPPING_INDEX in MAPPING_INFO.

1.6.3.2.3 Get specific information for a PAGE

Category Page switching, optional
Mnemonic GET_PAGE_INFO

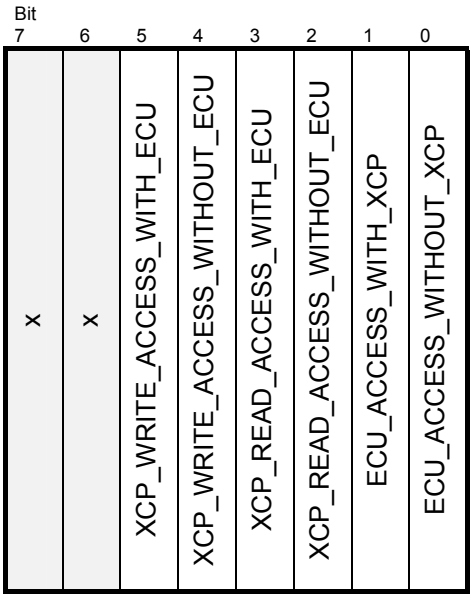
Position	Type	Description
0	BYTE	Command Code = 0xE7
1	BYTE	Reserved
2	BYTE	SEGMENT_NUMBER [0,1,..MAX_SEGMENT-1]
3	BYTE	PAGE_NUMBER [0,1,..MAX_PAGE-1]

GET_PAGE_INFO returns information on a specific PAGE.
If the specified PAGE is not available, ERR_OUT_OF_RANGE will be returned.

Positive response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	PAGE_PROPERTIES
2	BYTE	INIT_SEGMENT [0,1,..MAX_SEGMENT-1] SEGMENT that initializes this PAGE

The PAGE_PROPERTIES parameter is a bit mask described below:



The ECU_ACCESS_x flags indicate whether and how the ECU can access this page.
If the ECU can access this PAGE, the ECU_ACCESS_x flags indicate whether the ECU can access this PAGE only if the *X*CP master does NOT access this PAGE at the same time, only if the *X*CP master accesses this page at the same time, or the ECU doesn't care whether the *X*CP master accesses this page at the same time or not.

Bit 1	Bit 0	ECU_ACCESS_TYPE
ECU_ACCESS_WITH_XCP	ECU_ACCESS_WITHOUT_XCP	
0	0	ECU access not allowed
0	1	without *X*CP only
1	0	with *X*CP only
1	1	don't care

The XCP_x_ACCESS_y flags indicate whether and how the *X*CP master can access this page. The flags make a distinction for the XCP_ACCESS_TYPE depending on the kind of access the *X*CP master can have on this page (READABLE and/or WRITEABLE).

Bit 3	Bit 2	XCP_READ_ACCESS_TYPE
XCP_READ_ACCESS_WITH_ECU	XCP_READ_ACCESS_WITHOUT_ECU	
0	0	*X*CP READ access not allowed
0	1	without ECU only
1	0	with ECU only
1	1	don't care

If the *X*CP master can access this PAGE, the XCP_READ_ACCESS_x flags indicate whether the *X*CP master can read from this PAGE only if the ECU does NOT access this PAGE at the same time, only if the ECU accesses this page at the same time, or the *X*CP master doesn't need to care whether the ECU accesses this page at the same time or not.

Bit 5	4	XCP_WRITE_ACCESS_TYPE
XCP_WRITE_ACCESS_WITH_ECU	XCP_WRITE_ACCESS_WITHOUT_ECU	
0	0	*X*CP WRITE access not allowed
0	1	without ECU only
1	0	with ECU only
1	1	don't care

If the *X*CP master can access this PAGE, the XCP_WRITE_ACCESS_x flags indicate whether the *X*CP master can write to this PAGE only if the ECU does NOT access this PAGE at the same time, only if the ECU accesses this page at the same time, or the *X*CP master doesn't need to care whether the ECU accesses this page at the same time or not.

PAGE 0 of the INIT_SEGMENT of a PAGE contains the initial data for this PAGE.

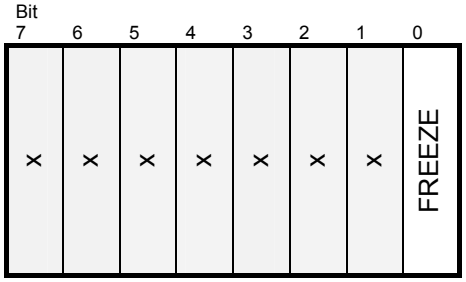
1.6.3.2.4 Set mode for a SEGMENT

Category Page switching, optional
Mnemonic SET_SEGMENT_MODE

Position	Type	Description
0	BYTE	Command Code = 0xE6
1	BYTE	Mode
2	BYTE	SEGMENT_NUMBER [0,1,..MAX_SEGMENT-1]

If the specified SEGMENT is not available, ERR_OUT_OF_RANGE will be returned.

The Mode parameter is a bit mask described below:



Flag	Description
FREEZE	0 = disable FREEZE Mode 1 = enable FREEZE Mode

The FREEZE flag selects the SEGMENT for freezing through STORE_CAL_REQ.

1.6.3.2.5 Get mode for a SEGMENT

Category Page switching, optional
Mnemonic GET_SEGMENT_MODE

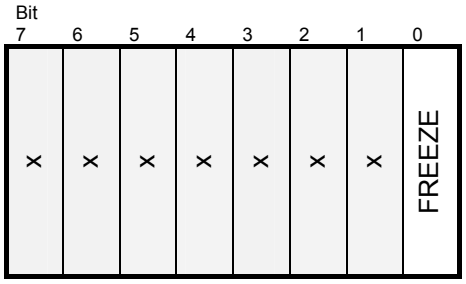
Position	Type	Description
0	BYTE	Command Code = 0xE5
1	BYTE	Reserved
2	BYTE	SEGMENT_NUMBER [0,1,..MAX_SEGMENT-1]

If the specified SEGMENT is not available, ERR_OUT_OF_RANGE will be returned.

Positive response:

Position	Type	Description
0	BYTE	Command Code = 0xFF
1	BYTE	reserved
2	BYTE	Mode

The Mode parameter is a bit mask described below:



1.6.3.2.6 Copy page

Category	Page switching, optional
Mnemonic	COPY_CAL_PAGE

This command forces the slave to copy one calibration page to another.

This command is only available if more than one calibration page is defined.

Position	Type	Description
0	BYTE	Command Code = 0xE4
1	BYTE	Logical data segment number source
2	BYTE	Logical data page number source
3	BYTE	Logical data segment number destination
4	BYTE	Logical data page number destination

In principal any page of any segment can be copied to any page of any segment. However, restrictions might be possible.

If calibration data page cannot be copied to the given destination, e.g. because the location of destination is a flash segment, an ERR_WRITE_PROTECTED will be returned. In this case Flash programming procedure has to be performed.

If the calibration data page is not available, an ERR_PAGE_NOT_VALID or ERR_SEGMENT_NOT_VALID will be returned.

1.6.4 Data Acquisition and Stimulation Commands (DAQ)

1.6.4.1 Static DAQ list configuration (stat)

1.6.4.1.1 Mandatory commands

1.6.4.1.1.1 Clear DAQ list configuration

Category Data acquisition and stimulation, static, mandatory
Mnemonic CLEAR_DAQ_LIST

Position	Type	Description
0	BYTE	Command Code = 0xE3
1	BYTE	reserved
2,3	WORD	DAQ_LIST_NUMBER [0,1..MAX_DAQ-1]

This command can be used for PREDEFINED and for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [0,1,..MAX_DAQ-1].

If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

CLEAR_DAQ_LIST clears the specified DAQ list. For a configurable DAQ list, all ODT entries will be reset to address=0, extension=0 and size=0 (if valid : bit_offset = 0xFF). For PREDEFINED and configurable DAQ lists, the running Data Transmission on this list will be stopped and all DAQ list states are reset.

1.6.4.1.1.2 Set pointer to ODT entry

Category Data acquisition and stimulation, static, mandatory
Mnemonic SET_DAQ_PTR

Position	Type	Description
0	BYTE	Command Code = 0xE2
1	BYTE	Reserved
2,3	WORD	DAQ_LIST_NUMBER [0,1,..MAX_DAQ-1]
4	BYTE	ODT_NUMBER [0,1,..MAX_ODT(DAQ list)-1]
5	BYTE	ODT_ENTRY_NUMBER [0,1,..MAX_ODT_ENTRIES(DAQ list)-1]

Initializes the DAQ list pointer for a subsequent operation with WRITE_DAQ or READ_DAQ.
If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

ODT_NUMBER is the relative ODT number within this DAQ list.

ODT_ENTRY_NUMBER is the relative ODT entry number within this ODT.

1.6.4.1.1.3 Write element in ODT entry

Category Data acquisition and stimulation, static, mandatory
Mnemonic WRITE_DAQ

Position	Type	Description
0	BYTE	Command Code = 0xE1
1	BYTE	BIT_OFFSET [0..31] Position of bit in 32-bit variable referenced by the address and extension below
2	BYTE	Size of DAQ element [AG] 0<= size <=MAX_ODT_ENTRY_SIZE_x
3	BYTE	Address extension of DAQ element
4..7	DWORD	Address of DAQ element

Writes one ODT entry to a DAQ list defined by the DAQ list pointer (see SET_DAQ_PTR).

WRITE_DAQ is only possible for elements in configurable DAQ lists. Therefore the DAQ_LIST_NUMBER used in the previous SET_DAQ_PTR has to be in the range [MIN_DAQ, MIN_DAQ+1,..MAX_DAQ-1]. Otherwise the slave will return an ERR_WRITE_PROTECTED as negative response upon WRITE_DAQ.

The BIT_OFFSET field allows the transmission of data stimulation elements that represent the status of a bit. For a MEASUREMENT that's in a DAQ list with DIRECTION = DAQ, the key word BIT_MASK describes the mask to be applied to the measured data to find out the status of a single bit. For a MEASUREMENT that's in a DAQ list with DIRECTION = STIM, the key word BIT_MASK describes the position of the bit that has to be stimulated. The Master has to transform the BIT_MASK to the BIT_OFFSET

e.g Bit7 → BIT_MASK = 0x80 → BIT_OFFSET = 0x07

When BIT_OFFSET = FF, the field can be ignored and the WRITE_DAQ applies to a normal data element with size expressed in AG. If the BIT_OFFSET is from 0x00 to 0x1F, the ODT entry describes an element that represents the status of a bit. In this case, the Size of DAQ element always has to be equal to the GRANULARITY_ODT_ENTRY_SIZE_x. If the value of this element = 0, the value for the bit = 0. If the value of the element > 0, the value for the bit = 1.

The size of an ODT entry has to fulfill the rules for granularity and maximum value. (ref. GET_DAQ_RESOLUTION_INFO).

The DAQ list pointer is auto post incremented to the next ODT entry within one and the same ODT. After writing to the last ODT entry of an ODT, the value of the DAQ pointer is undefined. The master has to make sure the correct position of the DAQ pointer when writing to the next ODT respectively the next DAQ list.

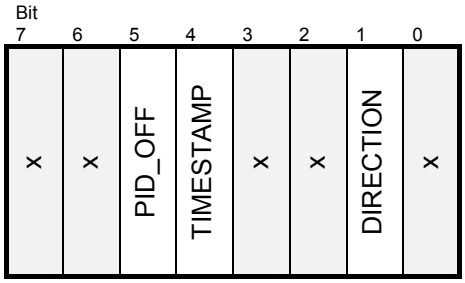
1.6.4.1.1.4 Set mode for DAQ list

Category Data acquisition and stimulation, static, mandatory
Mnemonic SET_DAQ_LIST_MODE

Position	Type	Description
0	BYTE	Command Code = 0xE0
1	BYTE	Mode
2,3	WORD	DAQ_LIST_NUMBER [0,1,..MAX_DAQ-1]
4,5	WORD	Event channel number [0,1,..MAX_EVENT_CHANNEL-1]
6	BYTE	Transmission rate prescaler (=>1)
7	BYTE	DAQ list priority (FF Highest)

This command can be used for PREDEFINED and for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [0,1,..MAX_DAQ-1].
If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

The Mode parameter is a bit mask described below:



Flag	Description
DIRECTION	0 = DAQ set to Data Acquisition Mode (Slave → Master) 1 = STIM set to Data Stimulation Mode (Master → Slave)
TIMESTAMP	0 = disable timestamp 1 = enable timestamp
PID_OFF	0 = transmit DTO with Identification Field 1 = transmit DTO without Identification Field

The DIRECTION flag sets the DAQ list into synchronized data acquisition or synchronized data stimulation mode.

The TIMESTAMP and PID_OFF flags can be used as well for DIRECTION = DAQ as for DIRECTION = STIM.

The TIMESTAMP flag sets the DAQ list into time stamped mode.

The TIMESTAMP_FIXED flag in TIMESTAMP_MODE at GET_DAQ_RESOLUTION_INFO indicates that the Master can not switch off the time stamp with SET_DAQ_LIST_MODE.
If the Master nevertheless tries to do so, the Slave will answer with an ERR_CMD_SYNTAX.

For DIRECTION = DAQ, time stamped mode means that the slave device transmits the current value of its clock in the first ODT of the DAQ cycle.
For DIRECTION = STIM, time stamped mode means that the master device first receives a time stamped DTO(DAQ) from the slave and then echoes this current value of the slave device's clock in the first ODT of the DAQ cycle. In this way the "time stamp" can be used as a counter that gives the slave the possibility to check whether DTO(DAQ) and CTO(STIM) belong functionally together.

The PID_OFF flag turns of the transmission of the Identification Field in each DTO packet. Turning off the transmission of the Identification Field is only allowed if the Identification Field Type is "absolute ODT number". If the Identification Field is not transferred in the *X*CP Packet, the unambiguous identification has to be done on the level of the Transport Layer. This can be done e.g. on CAN with separate CAN-Ids for each DAQ list and only one ODT for each DAQ list. In this case turning off the Identification Field would allow the transmission of 8 byte signals on CAN.

The Event Channel Number specifies the generic signal source that effectively determines the data transmission timing.

To allow reduction of the desired transmission rate, a transmission rate prescaler may be applied to the DAQ lists. Without reduction, the prescaler value must equal 1. For reduction, the prescaler has to be greater than 1.The use of a prescaler is only used for DAQ lists with DIRECTION = DAQ.

The DAQ list priority specifies the priority of this DAQ list if this DAQ list is processed together with other DAQ lists. The slave device driver may use this information to prioritize the transmission of data packets. DAQ list priority = 0 means that the slave may buffer the data and process them in a background task. DAQ list priority > 0 means that the slave has to process the data as fast as possible within the current raster. The DAQ-list with DAQ list priority = FF has the highest priority

If the ECU doesn't support the prioritization of DAQ lists , a DAQ list priority > 0 is not allowed and will be indicated by returning ERR_OUT_OF_RANGE.

1.6.4.1.1.5 Get mode from DAQ list

Category Data acquisition and stimulation, static, mandatory
Mnemonic GET_DAQ_LIST_MODE

Position	Type	Description
0	BYTE	Command Code = 0xDF
1	BYTE	Reserved
2,3	WORD	DAQ_LIST_NUMBER [0,1,..MAX_DAQ-1]

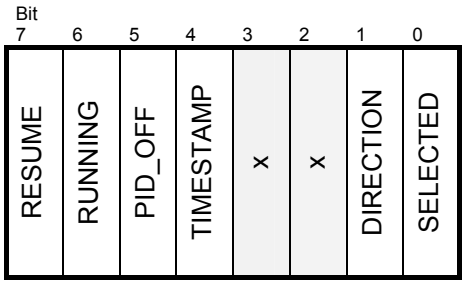
Returns information on the current mode of the specified DAQ list. This command can be used for PREDEFINED and for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [0,1,..MAX_DAQ-1].

If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Current Mode
2,3	WORD	Reserved
4,5	WORD	Current Event Channel Number
6	BYTE	Current Prescaler
7	BYTE	Current DAQ list Priority

The Current Mode parameter is a bit mask described below:



Flag	Description
SELECTED	0 = DAQ list not selected 1 = DAQ list selected
DIRECTION	0 = DAQ Data Acquisition Mode (Slave → Master) is set 1 = STIM Data Stimulation Mode (Master → Slave) is set
TIMESTAMP	0 = timestamp is disabled 1 = timestamp is enabled
PID_OFF	0 = DTO is transmitted with Identification Field 1 = DTO is transmitted without Identification Field
RUNNING	0 = DAQ list is inactive 1 = DAQ list is active
RESUME	0 = this DAQ list is not part of a configuration used in RESUME mode 1 = this DAQ list is part of a configuration used in RESUME mode

The SELECTED flag indicates that the DAQ list has been selected by a previous START_STOP_DAQ_LIST(select). If the next command is START_STOP_SYNCH, this will start/stop this DAQ list synchronously with other DAQ lists that are in the mode SELECTED. If the next command is SET_REQUEST, this will make the DAQ list to be part of a configuration that afterwards will be cleared or stored into non-volatile memory.

The DIRECTION flag indicates whether this DAQ list is configured for synchronous data acquisition or stimulation.

The RUNNING flag indicates that the DAQ list has been started actively by the master by a START_STOP_DAQ_LIST or START_STOP_SYNCH, or that the slave being in RESUME mode started the DAQ list automatically.

The RESUME flag indicates that this DAQ list is part of a configuration used in RESUME mode.

1.6.4.1.1.6 Start /stop/select DAQ list

Category Data acquisition and stimulation, static, mandatory
Mnemonic START_STOP_DAQ_LIST

Position	Type	Description
0	BYTE	Command Code = 0xDE
1	BYTE	Mode 00 = stop 01 = start 02 = select
2,3	WORD	DAQ_LIST_NUMBER [0,1,..MAX_DAQ-1]

This command can be used for PREDEFINED and for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [0,1,..MAX_DAQ-1].
If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

This command is used to start, stop or to prepare a synchronized start of the specified DAQ_LIST_NUMBER.

The mode parameter allows to start or stop this specific DAQ list.

The select mode configures the DAQ list with the provided parameters but does not start the data transmission of the specified list. This mode is used for a synchronized start/stop of all configured DAQ lists (ref. START_STOP_SYNCH) or for preparing the slave for RESUME mode (ref. SET_REQUEST).

The slave has to reset the SELECTED flag in the mode at GET_DAQ_LIST_MODE as soon as the related START_STOP_SYNCH or SET_REQUEST have been acknowledged.

If at least one DAQ list has been started, the slave device is in data transfer mode. The GET_STATUS command will return the DAQ_RUNNING status bit set.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	FIRST_PID

If the DTO Packets have an Identification Field Type “absolute ODT number”, FIRST_PID is the absolute ODT number in the DTO Packet of the first ODT transferred by this DAQ list.

The absolute ODT number for any other ODT can be determined by:

Absolute_ODT_number(ODT i in DAQ list j)
= FIRST_PID(DAQ list j) + relative_ODT_NUMBER(ODT i)

If the DTO Packets have an Identification Field Type “relative ODT number and absolute DAQ list number”, FIRST_PID can be ignored.

1.6.4.1.1.7 Start/stop DAQ lists (synchronously)

Category Data acquisition and stimulation, static, mandatory
Mnemonic START_STOP_SYNCH

Position	Type	Description
0	BYTE	Command Code = 0xDD
1	BYTE	Mode 00 = stop all 01 = start selected 02 = stop selected

This command is used to perform a synchronized start/stop of the transmission of DAQ lists.

The parameter Mode indicates the action and whether the command applies to all DAQ lists or to the selected ones only (previously configured with START_STOP_DAQ_LIST(select)). The slave device software has to reset the mode SELECTED of a DAQ list after successful execution of a START_STOP_SYNCH.

1.6.4.1.2 Optional commands

1.6.4.1.2.1 Get DAQ clock from slave

Category Data acquisition and stimulation, static, optional
Mnemonic GET_DAQ_CLOCK

Position	Type	Description
0	BYTE	Command Code = 0xDC

This command is used to synchronize the free running data acquisition clock of the slave device with the data acquisition clock in the master device. It is optional, if the slave device does not support timestamped data acquisition.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1..3	BYTE	Reserved
4..7	DWORD	Receive Timestamp

The returned receive timestamp has the format specified by the GET_DAQ_RESOLUTION_INFO command. It contains the current value of the data acquisition clock, when the GET_DAQ_CLOCK command packet has been received. The accuracy of the time synchronization between the master and the slave device is depending on the accuracy of this value.

On CAN based systems, the master device would be able to determine when the GET_DAQ_CLOCK command packet has been transmitted. This value corresponds to the point in time, when it has been received in the slave device. Based on the returned timestamp, the master device can calculate the time offset between the master and the slave device clock.

Compensating the time drift between the master and the slave device clocks is in the responsibility of the master device

1.6.4.1.2.2 Read element from ODT entry

Category Data acquisition and stimulation, static, optional
Mnemonic READ_DAQ

Position	Type	Description
0	BYTE	Command Code = 0xDB

Reads one ODT entry of a DAQ list defined by the DAQ list pointer. The DAQ list pointer is auto post incremented within one and the same ODT (See WRITE_DAQ).

READ_DAQ is possible for elements in PREDEFINED and configurable DAQ lists. Therefore the DAQ_LIST_NUMBER used in the previous SET_DAQ_PTR can be in the range [0,1,..MAX_DAQ-1].

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	BIT_OFFSET [0..31] Position of bit in 32-bit variable referenced by the address and extension below
2	BYTE	Size of DAQ element [AG] 0<= size <=MAX_ODT_ENTRY_SIZE_x
3	BYTE	Address extension of DAQ element
4..7	DWORD	Address of DAQ element

The size of an ODT entry has to fulfill the rules for granularity and maximum value. (ref. GET_DAQ_RESOLUTION_INFO).

1.6.4.1.2.3 Get general information on DAQ processor

Category Data acquisition and stimulation, static, optional
Mnemonic GET_DAQ_PROCESSOR_INFO

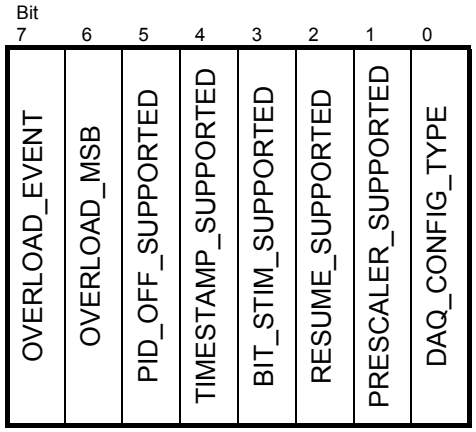
Position	Type	Description
0	BYTE	Command Code = 0xDA

This command returns general information on DAQ lists.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	DAQ_PROPERTIES General properties of DAQ lists
2,3	WORD	MAX_DAQ Total number of available DAQ lists
4,5	WORD	MAX_EVENT_CHANNEL Total number of available event channels
6	BYTE	MIN_DAQ Total number of predefined DAQ lists
7	BYTE	DAQ_KEY_BYTE

The DAQ_PROPERTIES parameter is a bit mask described below:



Flag	Description
DAQ_CONFIG_TYPE	0 = static DAQ list configuration 1 = dynamic DAQ list configuration
PRESCALER_SUPPORTED	0 = Prescaler not supported 1 = prescaler supported
RESUME_SUPPORTED	0 = DAQ lists can not be set to RESUME mode. 1 = DAQ lists can be set to RESUME mode.
BIT_STIM_SUPPORTED	0 = bitwise data stimulation not supported 1 = bitwise data stimulation supported
TIMESTAMP_SUPPORTED	0 = time stamped mode not supported 1 = time stamped mode supported
PID_OFF_SUPPORTED	0 = Identification Field can not be switched off 1 = Identification Field may be switched off

The DAQ_CONFIG_TYPE flag indicates whether the DAQ lists that are not PREDEFINED shall be configured statically or dynamically.

The PRESCALER_SUPPORTED flag indicates that all DAQ lists support the prescaler for reducing the transmission period.

The RESUME_SUPPORTED flag indicates that all DAQ lists can be put in RESUME mode.

The BIT_STIM_SUPPORTED flag indicates whether bitwise data stimulation through BIT_OFFSET in WRITE_DAQ is supported.

The TIMESTAMP_SUPPORTED flag indicates whether the slave supports time stamped data acquisition and stimulation. If the slave doesn't support a time stamped mode, the parameters TIMESTAMP_MODE and TIMESTAMP_TICKS (GET_DAQ_RESOLUTION_INFO) are invalid.

The OVERLOAD_MSB and OVERLOAD_EVENT flags indicate the used overrun indication method:

Bit 7	6	
OVERLOAD_EVENT	OVERLOAD_MSB	Overload indication type
0	0	No overload indication
0	1	overload indication in MSB of PID
1	0	overload indication by Event Packet
1	1	not allowed

For indicating an overrun situation, the slave may set the Most Significant Bit (MSB) of the PID of the next successfully transmitted packet. When the MSB of the PID is used, the maximum number of (absolute or relative) ODT numbers is limited and has to be in the range :

0x00<= ODT_NUMBER(DAQ with overrun_msb)<=0x7C

Alternatively the slave may transmit an „EV_DAQ_OVERLOAD„ event packet. The slave must take care not to overflow another cycle with this additional packet.

MAX_DAQ is the total number of DAQ lists available in the slave device. It includes the predefined DAQ lists that are not configurable (indicated with PREDEFINED at GET_DAQ_LIST_INFO) and the ones that are configurable. If DAQ_CONFIG_TYPE = dynamic, MAX_DAQ equals MIN_DAQ+DAQ_COUNT.

MIN_DAQ is the number of predefined DAQ lists. For predefined DAQ lists, DAQ_LIST_NUMBER is in the range [0,1,..MIN_DAQ-1].

DAQ_COUNT is the number of dynamically allocated DAQ lists.

MAX_DAQ-MIN_DAQ is the number of configurable DAQ lists. For configurable DAQ lists, DAQ_LIST_NUMBER is in the range [MIN_DAQ,MIN_DAQ+1,..MAX_DAQ-1].

MAX_EVENT_CHANNEL is the number of available event channels.

MAX_EVENT_CHANNEL = 0x00 means that the number of events in the slave is unknown.

The DAQ_KEY_BYTE parameter is a bit mask described below:

Bit 7	6	5	4	3	2	1	0
Identification_Field_Type_1	Identification_Field_Type_0	Address_Extension_DAQ	Address_Extension_ODT	Optimisation_Type_3	Optimisation_Type_2	Optimisation_Type_1	Optimisation_Type_0

The Optimisation_Type is defined as follows:

Bit 3	2	1	0	Optimisation Type
Optimisation_Type_3	Optimisation_Type_2	Optimisation_Type_1	Optimisation_Type_0	
0	0	0	0	OM_DEFAULT
0	0	0	1	OM_ODT_TYPE_16
0	0	1	0	OM_ODT_TYPE_32
0	0	1	1	OM_ODT_TYPE_64
0	1	0	0	OM_ODT_TYPE_ALIGNMENT
0	1	0	1	OM_MAX_ENTRY_SIZE

The Optimisation_Type flags indicate the Type of Optimisation Method the master preferably should use.

The Address_Extension is defined as follows:

Bit 5	4	
Address_Extension_DAQ	Address_Extension_ODT	Address_Extension Type
0	0	address extension can be different within one and the same ODT
0	1	address extension to be the same for all entries within one ODT
1	0	Not allowed
1	1	address extension to be the same for all entries within one DAQ

The ADDR_EXTENSION flag indicates whether the address extension of all entries within one ODT or within one DAQ must be the same.

The Identification_Field_Type is defined as follows:

Bit 7	6	
Identification_Field_Type_1	Identification_Field_Type_0	Identification Field Type
0	0	Absolute ODT number
0	1	Relative ODT number, absolute DAQ list number (BYTE)
1	0	Relative ODT number, absolute DAQ list number (WORD)
1	1	Relative ODT number, absolute DAQ list number (WORD, aligned)

The Identification_Field_Type flags indicate the Type of Identification Field the slave will use when transferring DAQ Packets to the master. The master has to use the same Type of Identification Field when transferring STIM Packets to the slave.

The PID_OFF_SUPPORTED flag in DAQ_PROPERTIES indicates that transfer of DTO Packets without Identification Field is possible.

Turning off the transfer of the Identification Field is only allowed if the Identification Field Type is “absolute ODT number”.

1.6.4.1.2.4 Get general information on DAQ processing resolution

Category Data acquisition and stimulation, static, optional
Mnemonic GET_DAQ_RESOLUTION_INFO

Position	Type	Description
0	BYTE	Command Code = 0xD9

This command returns information on the resolution of DAQ lists.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	GRANULARITY_ODT_ENTRY_SIZE_DAQ Granularity for size of ODT entry (DIRECTION = DAQ)
2	BYTE	MAX_ODT_ENTRY_SIZE_DAQ Maximum size of ODT entry (DIRECTION = DAQ)
3	BYTE	GRANULARITY_ODT_ENTRY_SIZE_STIM Granularity for size of ODT entry (DIRECTION = STIM)
4	BYTE	MAX_ODT_ENTRY_SIZE_STIM Maximum size of ODT entry (DIRECTION = STIM)
5	BYTE	TIMESTAMP_MODE Timestamp unit and size
6,7	WORD	TIMESTAMP_TICKS Timestamp ticks per unit

The possible values for GRANULARITY_ODT_ENTRY_SIZE_x are {1,2,4,8}.

For the address of the element described by an ODT entry, the following has to be fulfilled :

Address mod GRANULARITY_ODT_ENTRY_SIZE_x = 0

For every size of the element described by an ODT entry, the following has to be fulfilled :

SizeOf(element described by ODT entry) mod GRANULARITY_ODT_ENTRY_SIZE_x = 0

The MAX_ODT_ENTRY_SIZE_x parameters indicate the upper limits for the size of the element described by an ODT entry.
For every size of the element described by an ODT entry the following has to be fulfilled :

SizeOf(element described by ODT entry) <= MAX_ODT_ENTRY_SIZE_x

If the slave doesn't support a time stamped mode (no TIMESTAMP_SUPPORTED in GET_DAQ_PROCESSOR_INFO), the parameters TIMESTAMP_MODE and TIMESTAMP_TICKS are invalid.
If the slave device supports a time stamped mode, TIMESTAMP_MODE and TIMESTAMP_TICKS contain information on the resolution of the data acquisition clock. The data acquisition clock is a free running counter, which is never reset or modified.

The TIMESTAMP_MODE parameter is a bit mask described below:

Bit 7	6	5	4	3	2	1	0
Unit_3	Unit_2	Unit_1	Unit_0	TIMESTAMP_FIXED	Size_2	Size_1	Size_0

Size is defined as follows:

Bit 2	1	0	Timestamp size [bytes]
Size_2	Size_1	Size_0	
0	0	0	No time stamp
0	0	1	1
0	1	0	2
0	1	1	Not allowed
1	0	0	4

The TIMESTAMP_FIXED flag indicates that the Slave always will send DTO Packets in time stamped mode.

Unit is defined as follows:

Bit 7	6	5	4	Timestamp unit
Unit_3	Unit_2	Unit_1	Unit_0	
0	0	0	0	DAQ_TIMESTAMP_UNIT_1NS 1 NS = 1 nanosecond = 10^{-9} second
0	0	0	1	DAQ_TIMESTAMP_UNIT_10NS
0	0	1	0	DAQ_TIMESTAMP_UNIT_100NS
0	0	1	1	DAQ_TIMESTAMP_UNIT_1US 1 US = 1 microsecond = 10^{-6} second
0	1	0	0	DAQ_TIMESTAMP_UNIT_10US
0	1	0	1	DAQ_TIMESTAMP_UNIT_100US
0	1	1	0	DAQ_TIMESTAMP_UNIT_1MS 1 MS = 1 millisecond = 10^{-3} second
0	1	1	1	DAQ_TIMESTAMP_UNIT_10MS
1	0	0	0	DAQ_TIMESTAMP_UNIT_100MS
1	0	0	1	DAQ_TIMESTAMP_UNIT_1S 1 S = 1 second = 1 second

The timestamp will increment by TIMESTAMP_TICKS per unit and wrap around if an overflow occurs.

1.6.4.1.2.5 Get specific information for a DAQ list

Category Data acquisition and stimulation, static, optional
Mnemonic GET_DAQ_LIST_INFO

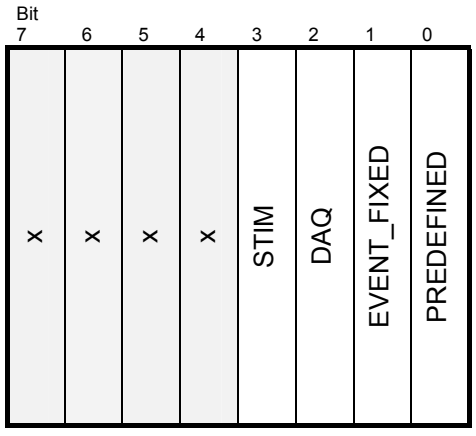
Position	Type	Description
0	BYTE	Command Code = 0xD8
1	BYTE	reserved
2,3	WORD	DAQ_LIST_NUMBER [0,1,...,MAX_DAQ-1]

GET_DAQ_LIST_INFO returns information on a specific DAQ list.
This command can be used for PREDEFINED and for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [0,1,..MAX_DAQ-1].
If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	DAQ_LIST_PROPERTIES Specific properties for this DAQ list
2	BYTE	MAX_ODT Number of ODTs in this DAQ list
3	BYTE	MAX_ODT_ENTRIES Maximum number of entries in an ODT
4,5	WORD	FIXED_EVENT Number of the fixed event channel for this DAQ list

The DAQ_LIST_PROPERTIES parameter is a bit mask described below:



Flag	Description
PREDEFINED	0 = DAQ list configuration can be changed 1 = DAQ list configuration is fixed
EVENT_FIXED	0 = Event Channel can be changed 1 = Event Channel is fixed

The PREDEFINED flag indicates that the configuration of this DAQ list can not be changed.

The DAQ list is predefined and fixed in the slave device's memory.

The EVENT_FIXED flag indicates that the Event Channel for this DAQ list can not be changed.

The DAQ and STIM flags indicate which DIRECTION can be used for this DAQ list

3	2	
STIM	DAQ	DAQ_LIST_TYPE
0	0	Not allowed
0	1	only DIRECTION = DAQ supported
1	0	only DIRECTION = STIM supported
1	1	both DIRECTIONS supported (but not simultaneously)

If DAQ lists are configured statically, MAX_ODT specifies the number of ODTs for this DAQ list and MAX_ODT_ENTRIES specifies the number of ODT entries in each ODT.

FIXED_EVENT indicates the number of the fixed event channel to be used for this DAQ list.

1.6.4.1.2.6 Get specific information for an event channel

Category Data acquisition and stimulation, static, optional
Mnemonic GET_DAQ_EVENT_INFO

Position	Type	Description
0	BYTE	Command Code = 0xD7
1	BYTE	Reserved
2,3	WORD	Event channel number [0,1,..MAX_EVENT_CHANNEL-1]

GET_DAQ_EVENT_INFO returns information on a specific event channel. A number in a range from 0 to MAX_EVENT_CHANNEL-1 addresses the event channel. If the specified event channel is not available, ERR_OUT_OF_RANGE will be returned.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	DAQ_EVENT_PROPERTIES Specific properties for this event channel
2	BYTE	MAX_DAQ_LIST [0,1,2,..255] maximum number of DAQ lists in this event channel
3	BYTE	Event channel name length in bytes 0 – If not available
4	BYTE	Event channel time cycle 0 – Not cyclic
5	BYTE	Event channel time unit 0 – Not cyclic
6	BYTE	Event channel priority (FF highest)

The DAQ_EVENT_PROPERTIES parameter is a bit mask described below:

Bit 7	6	5	4	3	2	1	0
x	x	x	x	STIM	DAQ	x	x

The DAQ and STIM flags indicate what kind of DAQ list can be allocated to this event channel :

3 STIM	2 DAQ	EVENT_CHANNEL_TYPE
0	0	Not allowed
0	1	only DAQ lists with DIRECTION = DAQ supported
1	0	only DAQ lists with DIRECTION = STIM supported
1	1	both kind of DAQ lists (simultaneously)

MAX_DAQ_LIST indicates the maximum number of DAQ lists that can be allocated to this event channel. MAX_DAQ_LIST = 0x00 means this event is available but currently can't be used. MAX_DAQ_LIST = 0xFF means there's no limitation.

This command automatically sets the Memory Transfer Address (MTA) to the location from which the master device may upload the event channel name as ASCII text, using one or more UPLOAD commands. The event channel name length specifies the number of ASCII bytes in the name. There must be no 0 termination.

The event channel time cycle indicates with what sampling period the slave processes this event channel.

See timestamp unit for the definition of event channel time unit.

The event channel priority specifies the priority of this event channel when the slave processes the different event channels. This prioritization is a fixed attribute of the slave and therefore read-only. The event channel with event channel priority = FF has the highest priority

1.6.4.2 Dynamic DAQ List Configuration (dyn)

1.6.4.2.1 Optional commands

1.6.4.2.1.1 Clear dynamic DAQ configuration

Category Data acquisition and stimulation, dynamic, optional
Mnemonic FREE_DAQ

Position	Type	Description
0	BYTE	Command Code = 0xD6

This command clears all DAQ lists and frees all dynamically allocated DAQ lists, ODTs and ODT entries.

At the start of a dynamic DAQ list configuration sequence, the master always first has to send a FREE_DAQ.

1.6.4.2.1.2 Allocate DAQ lists

Category Data acquisition and stimulation, dynamic, optional
Mnemonic ALLOC_DAQ

Position	Type	Description
0	BYTE	Command Code = 0xD5
1	BYTE	reserved
2,3	WORD	DAQ_COUNT number of DAQ lists to be allocated

This command allocates a number of DAQ lists for this *X*CP slave device.

If there's not enough memory available to allocate the requested DAQ lists an ERR_MEMORY_OVERFLOW will be returned as negative response.

The master has to use ALLOC_DAQ in a defined sequence together with FREE_DAQ, ALLOC_ODT and ALLOC_ODT_ENTRY. If the master sends an ALLOC_DAQ directly after an ALLOC_ODT without a FREE_DAQ in between, the slave returns an ERR_SEQUENCE as negative response.

If the master sends an ALLOC_DAQ directly after an ALLOC_ODT_ENTRY without a FREE_DAQ in between, the slave returns an ERR_SEQUENCE as negative response.

1.6.4.2.1.3 Allocate ODTs to a DAQ list

Category Data acquisition and stimulation, dynamic, optional
Mnemonic ALLOC_ODT

Position	Type	Description
0	BYTE	Command Code = 0xD4
1	BYTE	Reserved
2,3	WORD	DAQ_LIST_NUMBER [MIN_DAQ, MIN_DAQ+1,..MIN_DAQ+DAQ_COUNT-1]
4	BYTE	ODT_COUNT number of ODTs to be assigned to DAQ list

This command allocates a number of ODTs and assigns them to the specified DAQ list.

This command can only be used for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [MIN_DAQ, MIN_DAQ+1,..MIN_DAQ+DAQ_COUNT-1].

If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

If there's not enough memory available to allocate the requested ODTs an ERR_MEMORY_OVERFLOW will be returned as negative response.

The master has to use ALLOC_ODT in a defined sequence together with FREE_DAQ, ALLOC_DAQ and ALLOC_ODT_ENTRY. If the master sends an ALLOC_ODT directly after a FREE_DAQ without an ALLOC_DAQ in between, the slave returns an ERR_SEQUENCE as negative response.

If the master sends an ALLOC_ODT directly after an ALLOC_ODT_ENTRY without a FREE_DAQ in between, the slave returns an ERR_SEQUENCE as negative response.

1.6.4.2.1.4 Allocate ODT entries to an ODT

Category Data acquisition and stimulation, dynamic, optional
Mnemonic ALLOC_ODT_ENTRY

Position	Type	Description
0	BYTE	Command Code = 0xD3
1	BYTE	Reserved
2,3	WORD	DAQ_LIST_NUMBER [MIN_DAQ, MIN_DAQ+1,..MIN_DAQ+DAQ_COUNT-1]
4	BYTE	ODT_NUMBER [0,1,..ODT_COUNT(DAQ list)-1]
5	BYTE	ODT_ENTRIES_COUNT number of ODT entries to be assigned to ODT

This command allocates a number of ODT entries and assigns them to the specific ODT in this specific DAQ list.

This command can only be used for configurable DAQ lists, so the range for DAQ_LIST_NUMBER is [MIN_DAQ, MIN_DAQ+1,..MIN_DAQ+DAQ_COUNT-1].

If the specified list is not available, ERR_OUT_OF_RANGE will be returned.

ODT_NUMBER is the relative ODT number within this DAQ list.

If there's not enough memory available to allocate the requested ODT entries an ERR_MEMORY_OVERFLOW will be returned as negative response.

The master has to use ALLOC_ODT_ENTRY in a defined sequence together with FREE_DAQ and ALLOC_ODT. If the master sends an ALLOC_ODT_ENTRY directly after a FREE_DAQ without an ALLOC_DAQ in between, the slave returns an ERR_SEQUENCE as negative response.

If the master sends an ALLOC_ODT_ENTRY directly after an ALLOC_DAQ without an ALLOC_ODT in between, the slave returns an ERR_SEQUENCE as negative response.

1.6.5 Non-volatile Memory Programming (PGM)

1.6.5.1 Mandatory commands

1.6.5.1.1 Indicate the beginning of a programming sequence

Category Non-volatile memory programming, mandatory
Mnemonic PROGRAM_START

Position	Type	Description
0	BYTE	Command Code = 0xD2

This command is used to indicate the begin of a non-volatile memory programming sequence. If the slave device is not in a state which permits programming, a ERR_GENERIC will be returned. The memory programming commands PROGRAM_CLEAR, PROGRAM, PROGRAM_MAX or PROGRAM_NEXT are not allowed, until the PROGRAM_START command has been successfully executed. The end of a non-volatile memory programming sequence is indicated by a PROGRAM_RESET command.

Memory programming may have implementation specific preconditions (slave device in a secure physical state, additional code downloaded, ...) and the execution of other commands may be restricted during a programming sequence (data acquisition may not run, calibration may be restricted, ...). The following commands must always be available during a memory programming sequence:

```
SET_MTA
PROGRAM_CLEAR
PROGRAM
PROGRAM_MAX or PROGRAM_NEXT
```

The following commands are optional (for instance to verify memory contents):

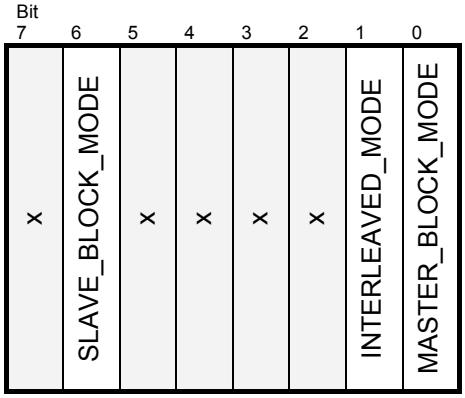
```
UPLOAD
BUILD_CHECKSUM
```

If non-volatile memory programming requires the download of additional code, the download has to be finished before the PROGRAM_START command is executed. The MTA must point to the entry point of the downloaded routine.

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Reserved
2	BYTE	COMM_MODE_PGM
3	BYTE	MAX_CTO_PGM [BYTES] Maximum CTO size for PGM
4	BYTE	MAX_BS_PGM
5	BYTE	MIN_ST_PGM
6	BYTE	QUEUE_SIZE_PGM

The COMM_MODE_PGM parameter is a bit mask described below :



The MASTER_BLOCK_MODE flag indicates whether the Master Block Mode is available during Programming.

The INTERLEAVED_MODE flag indicates whether the Interleaved Mode is available during Programming.

The SLAVE_BLOCK_MODE flag indicates whether the Slave Block Mode is available during Programming.

The communication parameters MAX_CTO, MAX_BS, MIN_ST and QUEUE_SIZE may change when the slave device is in memory programming mode. The new communication parameters MAX_CTO_PGM, MAX_BS_PGM, MIN_ST_PGM and QUEUE_SIZE_PGM are returned in the positive response.

1.6.5.1.2 Clear a part of non-volatile memory

Category Non-volatile memory programming, mandatory
Mnemonic PROGRAM_CLEAR

Position	Type	Description
0	BYTE	Command Code = 0xD1
1	BYTE	Mode
2,3	WORD	reserved
4..7	DWORD	clear range

This command is used to clear a part of non-volatile memory prior to reprogramming.
The work flow depends on mode byte

Mode Byte	Description
0x00	the absolute access mode is active (default)
0x01	the functional access mode is active

Absolute Access mode

Parameter	Description
MTA	The MTA points to the start of a memory sector inside the slave. Memory sectors are described in the ASAM MCD 2MC slave device description file. If multiple memory sectors shall be cleared in a certain sequence, the master device must repeat the PROGRAM_CLEAR service with a new MTA. In this case the master must keep the order information given by the Clear Sequence Number of the sectors.
Clear range	The Clear Range indicates the length of the memory part to be cleared. The PROGRAM_CLEAR service clears a complete sector or multiple sectors at once.

Functional Access mode

Parameter	Description
MTA	The MTA has no influence on the clearing functionality
clear range	This parameter should be interpreted bit after bit: basic use-cases: 0x00000001 : clear all the calibration data area(s) 0x00000002 : clear all the code area(s) (the boot area is not covered) 0x00000004 : clear the NVRAM area(s) 0x00000008 .. 0x00000080: reserved project specific use-cases: 0x00000100 ... 0xFFFFFF00 : user defined

Example

If the project divides the calibration area into different areas, it is useful to define the project specific higher nibble as follow:

0x00000100 : clear calibration data area 1
0x00000200 : clear calibration data area 2
0x00000400 : clear calibration data area 3
...

In this use case the different calibration areas can be reprogrammed without further information of the memory mapping and the flash organisation. These parameters must be specified in the project specific programming flow control.

1.6.5.1.3 Program a non-volatile memory segment

Category Non-volatile memory programming, mandatory
Mnemonic PROGRAM

Position	Type	Description
0	BYTE	Command Code = 0xD0
1	BYTE	Number of data elements [AG] [1..(MAX_CTO-2)/AG]
2..AG-1	BYTE	Used for alignment, only if AG >2
AG=1: 2..MAX_CTO-2 AG>1: AG MAX_CTO-AG	ELEMENT	Data elements

If ADDRESS_GRANULYRITY = DWORD, 2 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE. WORD or DWORD, depending upon AG.

This command is used to program data inside the slave. Depending on the access mode (defined by PROGRAM_FORMAT) 2 different concepts are supported.

The end of the memory segment is indicated, when the number of data elements is 0.

The end of the overall programming sequence is indicated by a PROGRAM_RESET command. The slave device will go to disconnected state. Usually a hardware reset of the slave device is executed.

This command may support block transfer similar to the commands DOWNLOAD and DOWNLOAD_NEXT.

Absolute Access mode

The data block of the specified length (size) contained in the CTO will be programmed into non-volatile memory, starting at the MTA. The MTA will be post-incremented by the number of data bytes.

If multiple memory sectors shall be programmed, the master device must keep the order information given in the IF_DATA description called Programming Sequence Number of the sector.

Functional Access mode

The data block of the specified length (size) contained in the CTO will be programmed into non-volatile memory. The ECU software knows the start address for the new flash content automatically. It depends on the PROGRAM_CLEAR command. The ECU expects the new flash content in one data stream and the assignment is done by the ECU automatically.

The MTA works as a Block Sequence Counter and it is counted inside the master and the server. The Block Sequence Counter allows an improved error handling in case a programming service fails during a sequence of multiple programming requests. The Block Sequence Counter of the server shall be initialized to one (1) when receiving a PROGRAM_FORMAT request message. This means that the first PROGRAM request message following the PROGRAM_FORMAT request message starts with a Block Sequence Counter of one (1). Its value is incremented by 1 for each subsequent data transfer request. At the maximum value the Block Sequence Counter rolls over and starts at 0x00 with the next data transfer request message.

1.6.5.1.4 Indicate the end of a programming sequence

Category Non-volatile memory programming, mandatory
Mnemonic PROGRAM_RESET

Position	Type	Description
0	BYTE	Command Code = 0xCF

This optional command indicates the end of a non-volatile memory programming sequence. It may or may not have a response. It either case, the slave device will go to the disconnected state.

This command may be used to force a slave device reset for other purposes.

1.6.5.2 Optional commands

1.6.5.2.1 Get general information on PGM processor

Category Programming, optional
Mnemonic GET_PGM_PROCESSOR_INFO

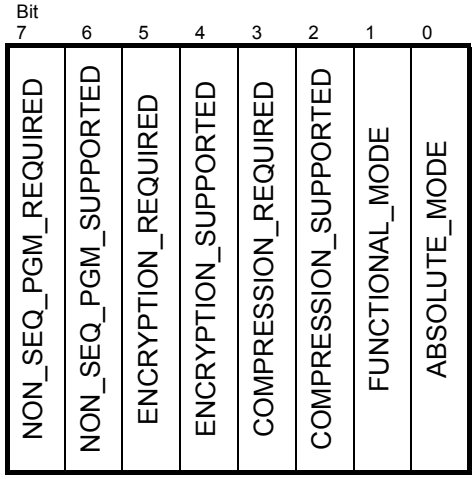
Position	Type	Description
0	BYTE	Command Code = 0xCE

This command returns general information on programming.

Positive response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	PGM_PROPERTIES General properties for programming
2	BYTE	MAX_SECTOR total number of available sectors

The PGM_PROPERTIES parameter is a bit mask described below:



The ABSOLUTE_MODE and FUNCTIONAL_MODE flags indicate the clear/programming mode that can be used

Bit 1	Bit 0	clear/programming mode
FUNCTIONAL_MODE	ABSOLUTE_MODE	
0	0	Not allowed
0	1	Only Absolute mode supported
1	0	Only Functional mode supported
1	1	Both modes supported

The COMPRESSION_x flags indicate which compression state of the incoming data the slave can process. The answer is a summary (OR operation) for all programmable segments and/or sectors.

Bit 3	Bit 2	compression
COMPRESSION_REQUIRED	COMPRESSION_SUPPORTED	
0	0	Not supported
0	1	supported
1	0	Supported and required
1	1	

The ENCRYPTION_x flags indicate which encryption state of the incoming data the slave can process. The answer is a summary (OR operation) for all programmable segments and/or sectors.

Bit 5	Bit 4	encryption
ENCRYPTION_REQUIRED	ENCRYPTION_SUPPORTED	
0	0	Not supported
0	1	supported
1	0	Supported and required
1	1	

The NON_SEQ_PGM_x flags indicate whether the slave can process different kind of sequence regarding the incoming data. The answer is a summary (OR operation) for all programmable segments and/or sectors.

Bit 7	Bit 6	non sequential programming
NON_SEQ_PGM_REQUIRED	NON_SEQ_PGM_SUPPORTED	
0	0	Not supported
0	1	supported
1	0	Supported and required
1	1	

MAX_SECTOR is the total number of sectors in the slave device

1.6.5.2.2 Get specific information for a SECTOR

Category Programming, optional
Mnemonic GET_SECTOR_INFO

Position	Type	Description
0	BYTE	Command Code = 0xCD
1	BYTE	Mode 0 = get start address for this SECTOR 1 = get length of this SECTOR
2	BYTE	SECTOR_NUMBER [0,1,..MAX_SECTOR-1]

GET_SECTOR_INFO returns information on a specific SECTOR.
If the specified SECTOR is not available, ERR_OUT_OF_RANGE will be returned.
This optional command is only helpful for the programming method 'absolute access mode'.

Positive response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Clear Sequence Number
2	BYTE	Program Sequence Number
3	BYTE	Programming method
4..7	DWORD	SECTOR_INFO mode = 0 : Start address for this SECTOR mode = 1 : Length of this SECTOR [AG]

The Clear Sequence Number and Program Sequence Number describe, in which subsequential order the master has to clear and program flash memory sectors. Each sequence number must be unique. Sectors, which do not have to be programmed, can be skipped in the programming flow control.

Example 1: In this example the memory must be cleared from small to great sector numbers and then reprogrammed from great to small sector numbers.

```
Sector      | Returned Value for Clear/Program Sequence Number
-------------------------------------------------------
Sector 0    |    0 / 5
Sector 1    |    1 / 4
Sector 2    |    2 / 3
```

Example 2: In this example the memory sectors must be alternately cleared and reprogrammed from small to great sector numbers.

```
Sector      | Returned Value for Clear/Program Sequence Number
-------------------------------------------------------
Sector 0    |    0 / 1
Sector 1    |    2 / 3
Sector 2    |    4 / 5
```

If Mode = 0 , this command returns the start address for this SECTOR in SECTOR_INFO.
If Mode = 1, this command returns the length of this SECTOR in SECTOR_INFO.

1.6.5.2.3 Prepare non-volatile memory programming

Category Non-volatile memory programming, optional
Mnemonic PROGRAM_PREPARE

Position	Type	Description
0	BYTE	Command Code = 0xCC
1	BYTE	Not used
2,3	WORD	Codesize [AG]

This optional command is used to indicate the begin of a code download as a precondition for non-volatile memory programming. The MTA points to the begin of the volatile memory location where the code will be stored. The parameter Codesize specifies the size of the code that will be downloaded. The download itself is done by using subsequent standard commands like SET_MTA and DOWNLOAD.

Codesize is expressed in BYTE, WORD or DWORD depending upon AG.

The slave device has to make sure that the target memory area is available and it is in a operational state which permits the download of code. If not, a ERR_GENERIC will be returned.

1.6.5.2.4 Set data format before programming

Category Non-volatile memory programming, optional
Mnemonic PROGRAM_FORMAT

Position	Type	Description
0	BYTE	Command Code = 0xCB
1	BYTE	Compression method
2	BYTE	Encryption method
3	BYTE	Programming method
4	BYTE	access method

This command describes the format of following, uninterrupted data transfer. The data format is set direct at begin of the programming sequence and is valid till end of this sequence. The sequence will be terminated by other commands e.g. SET_MTA.

If this command isn't transmitted at begin of a sequence, unmodified data and absolute address access method is supposed.

If modified data transmission is expected by the slave and no PROGRAM_FORMAT command is transmitted, the slave responds with ERR_SEQUENCE.
The reformatting methods are described as follows :

Parameter	Hex	Description
Compression method	0x00 0x80...0xFF	Data uncompressed (default) User defined
Encryption method	0x00 0x80...0xFF	Data not encrypted (default) User defined
Programming method	0x00 0x80...0xFF	Sequential Programming (default) User defined
Access method	0x00	Absolute Access Mode (default) The MTA uses physical addresses
	0x01	Functional Access Mode The MTA functions as a block sequence number of the new flash content file.
	0x80...0xFF	User defined
		It is possible to use different access modes for clearing and programming.

The master will not perform the reformatting. The master just is getting the values that identify the reformatting methods from the ASAM MCD-MC2 description file and passing them to the slave.

Affected Commands

PROGRAM, PROGRAM_MAX, PROGRAM_NEXT, SET_MTA

Example

...
SET_MTA *program code, encrypted*
PROGRAM_FORMAT
PROGRAM
PROGRAM_NEXT1..n
...

1.6.5.2.5 Program a non-volatile memory segment (Block Mode)

Category Non-volatile memory programming, optional
Mnemonic PROGRAM_NEXT

Position	Type	Description
0	BYTE	Command Code = 0xCA
1	BYTE	Number of data elements [AG] [1..(MAX_CTO-2)/AG]
2..AG-1	BYTE	Used for alignment, only if AG >2
AG=1: 2..MAX_CTO-2 AG>1: AG MAX_CTO-AG	ELEMENT	Data elements

If AG = DWORD, 2 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE, WORD or DWORD, depending upon AG.

This command is used to transmit consecutive data bytes for the PROGRAM command in block transfer mode.

Negative Response:

If the number of data elements does not match the expected value, the error code ERR_SEQUENCE will be returned. The negative response will contain the expected number of data elements.

Position	Type	Description
0	BYTE	Packet ID: 0xFE
1	BYTE	ERR_SEQUENCE
2	BYTE	Number of expected data elements

1.6.5.2.6 Program a non-volatile memory segment (fixed size)

Category Non-volatile memory programming, optional
Mnemonic PROGRAM_MAX

Position	Type	Description
0	BYTE	Command Code = 0xC9
1..AG-1	BYTE	Used for alignment, only if AG >1
AG..MAX_CTO-AG	ELEMENT	Data elements

Depending upon AG, 1 or 3 alignment bytes must be used in order to meet alignment requirements.

ELEMENT is BYTE, WORD or DWORD, depending upon AG.

The data block with the fixed length of MAX_CTO-1 elements contained in the CTO will be programmed into non-volatile memory, starting at the MTA. The MTA will be post-incremented by MAX_CTO-1.

This command does not support block transfer and it may not be used within a block transfer sequence.

1.6.5.2.7 Program Verify

Category Non-volatile memory programming, optional
Mnemonic PROGRAM_VERIFY

Position	Type	Description
0	BYTE	Command Code = 0xC8
1	BYTE	Verification mode 00 = request to start internal routine 01 = sending Verification Value
2,3	WORD	Verification Type
4..7	DWORD	Verification Value

With Verification Mode = 00 the master can request the slave to start internal test routines to check whether the new flash contents fits to the rest of the flash. Only the result is of interest.

With Verification Mode = 01, the master can tell the slave that he will be sending a Verification Value to the slave.

The definition of the Verification Mode is project specific. The master is getting the Verification Mode from the project specific programming flow control and passing it to the slave.

The tool needs no further information about the details of the project specific check routines. The *X*CP parameters allow a wide range of project specific adaptions.

The Verification Type is a bit mask described below:

Verification Type	Description
0x0001	calibration area(s) of the flash
0x0002	code area(s) of the flash
0x0004	complete flash content
0x0008 ... 0x0080	reserved
0x0100 ... 0xFF00	user defined

The Verification Type is specified in the project specific programming flow control. The master is getting this parameter and passing it to the slave.

The definition of the Verification Value is project specific and the use is defined in the project specific programming flow control.

1.7 Communication Error Handling

1.7.1 Definitions

1.7.1.1 Error

When the master sends a command CMD to the slave, no error occurs if the slave within a specified time answers with a positive response RES.

A Time-out Error occurs if the slave doesn't answer with any response within a specified time.
An Error Code Error occurs if the slave answers within a specified time with a negative response ERR.

1.7.1.2 Pre-action

When trying to recover from an Error, the master first has to perform a Pre-Action and then an Action.

The Pre-Action brings the slave in a well defined state that allows the master to perform the Action.

The *X*CP Protocol supports the following kind of Pre-Actions :

- Wait t_7
- SYNCH
- GET_SEED / UNLOCK
- SET_MTA
- SET_DAQ_PTR
- START_STOP_x
- Reinitialise DAQ

1.7.1.3 Action

With the Action, the master tries to recover from the Error State.
The *X*CP Protocol supports the following kind of Actions :

- Display error
- Retry other syntax
- Retry other parameter
- Use ASAM MCD 2MC Description File
- Use alternative
- Repeat 2 times
- Repeat ∞ times
- Restart session
- Terminate session

1.7.1.4 Error severity

Error and Event messages are classified according to their Severity Level.
The *X*CP Protocol defines the following Severity Levels :

Severity	Description
S0	Information
S1	Warning / Request
S2	Resolvable Error
S3	Fatal Error

The Severity Level gives the master information about a possible Transition in the State-machine and for deciding about an appropriate reaction upon the ERR or EV.

1.7.2 Time-out Handling

A Time-out Error occurs if the slave within a specified time doesn't answer with any response to a command sent from master to slave.

When sending a command, the master has to start a timer. For each command, the maximum value the timer can reach is given by the Time-Out Value t_x. If the master receives an answer before the timer reaches its maximal value, the master has to reset the timer. If the timer reaches its maximum value without the master receiving an answer from the slave, the master has to detect this as a Time-Out Error.

The *X*CP Protocol supports 6 different Time-Out Values t_1 till t_6.

The master can get the values for t_1 till t_6 from the ASAM MCD 2MC Description File.

The specific t_x for each command is indicated in the chapter "Error Handling Matrix".

1.7.2.1 Standard Communication Model

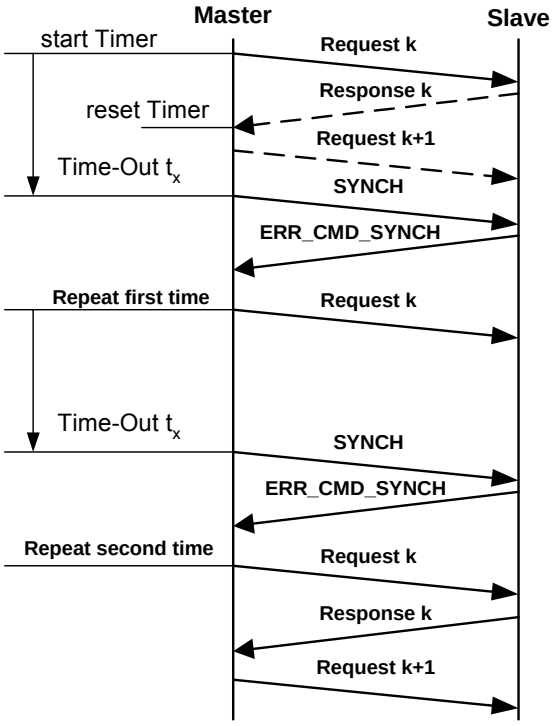


Diagram 24 : Time-out Handling in Standard Communication Model

If the Master detects a Time-out in the Standard Communication Model, the master has to perform the Pre-Action and Action. This sequence (pre-action, action) has to be tried 2 times.
If the master then still detects a Time-out Error, the master can decide about an appropriate reaction by himself.

In the usual case, the (pre-action, action) consists of a SYNCH command to re-synchronize command execution between master and slave followed by a repetition of the command. For some special commands, the pre-action brings the slave in a well defined state e.g. by sending again SET_MTA or SET_DAQ_PTR before repeating the command.

1.7.2.2 Block Communication Model

If the Master detects a Time-out in the Block Communication Model, the master has to perform the same Error Handling as for the Standard Communication Model.

In Master Block Transfer Mode, the master has to start the timer used for Time-out detection when sending the last frame of a block that builds a command.

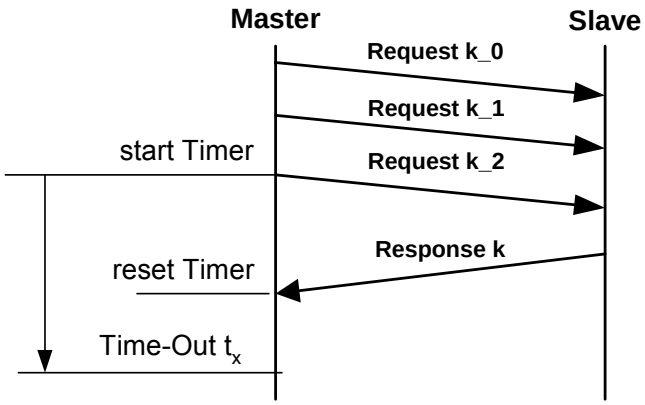


Diagram 25 : Time-out Handling in Master Block Transfer Mode

In Master Block Transfer Mode, the master has to use the same Time-Out Value t_x it uses when sending the same command in Standard Communication mode.

When repeating a command, the master always has to repeat the complete block that builds the command.

In Slave Block Transfer Mode, the master has to reset the timer used for Time-out detection when receiving the last frame of a block that builds a response.

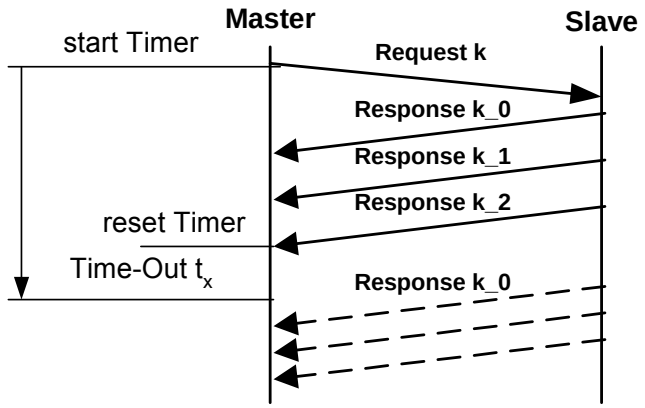


Diagram 26 : Time-out Handling in Slave Block Transfer Mode

In Slave Block Transfer Mode, the master has to use the same Time-Out Value t_x it uses when receiving the same response in Standard Communication mode.

1.7.2.3 Interleaved Communication Model

If the Master detects a Time-out in the Interleaved Communication Model, the master has to perform the same Error Handling as for the Standard Communication Model.

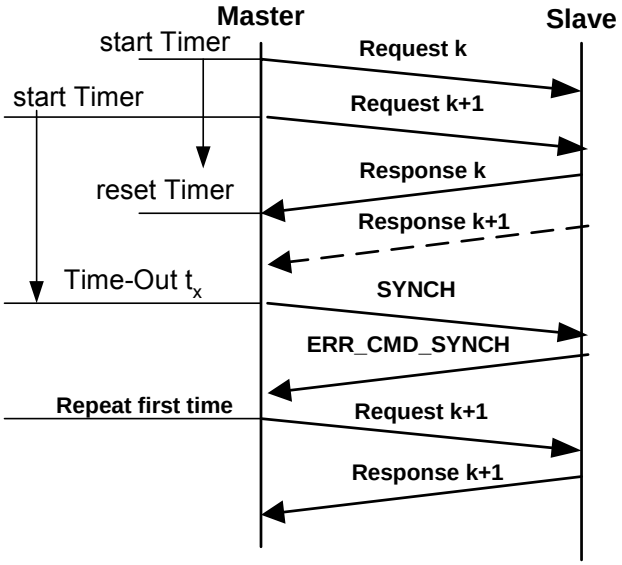


Diagram 27 : Time-out Handling in Interleaved Communication Model

1.7.2.4 Time-Out manipulation

The master gets the default values for t_1 till t_6 from the ASAM MCD 2MC Description File.
For special purposes, *X*CP allows to overrule these Time-Out Values.
With EV_CMD_PENDING, the slave can request the master to restart the time-out detection.

1.7.2.4.1 Overruling Time-Out values

For bypassing, it might be necessary to change the Time-Out Values used by the slave.

The setting of these values is done by standard calibration methods. No special *X*CP commands are needed for this.

1.7.2.4.2 Restarting Time-Out detection

With EV_CMD_PENDING, the slave can request the master to restart the time-out detection.

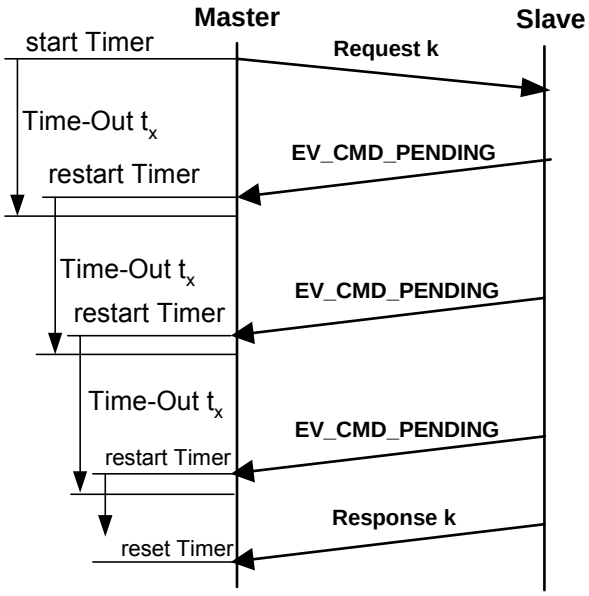


Diagram 28 : Restarting Time-out detection with EV_CMD_PENDING

The EV_CMD_PENDING allows the slave to inform the master that the request was correctly received and the parameters in the request are valid. However, the slave currently is not able of generating a response yet.

If the master receives an EV_CMD_PENDING from the slave, the master shall not repeat the request.

If the master receives an EV_CMD_PENDING from the slave, the master has to restart the timer used for time-out detection.

As soon as the slave has been able to process the request, it has to send a (positive or negative) response RES or ERR to the master.

1.7.3 Error Code Handling

An Error Code Error occurs if the slave answers within a specified time with a negative response ERR.

If the master sends a command which belongs to a not supported resource, the slave responds with an ERR_CMD_UNKNOWN.

If the master receives an ERR when sending a CMD, it has to check the “Error Handling Matrix” that for each CMD defines a specific “Pre-Action” and “Action” for a specific ERR.
If the master after performing the “Pre-Action” and “Action” still detects an Error Code Error, the master can decide about an appropriate reaction by himself.

If for a specific CMD, the specific ERR is not mentioned in the “Error Handling Matrix”, the master has to check the Severity of this ERR in the “Table of Error Codes” and decide about an appropriate reaction.

If an error occurs during a multi-command sequence, the master can decide about an appropriate reaction.

1.7.3.1 Table of Error Codes (ERR_*)

The Error packet codes in the table below may be sent as an asynchronous packet with PID 0xFE.

The Error code **0x00** is used for synchronization purposes (ref. description of SYNCH).

Error	Code	Description	Severity
ERR_CMD_SYNCH	0x00	Command processor synchronization.	S0

An Error code **ERR_* >= 0x01** is used for Error packets.

Error	Code	Description	Severity
ERR_CMD_BUSY	0x10	Command was not executed.	S2
ERR_DAQ_ACTIVE	0x11	Command rejected because DAQ is running.	S2
ERR_PGM_ACTIVE	0x12	Command rejected because PGM is running.	S2

Error	Code	Description	Severity
ERR_CMD_UNKNOWN	0x20	Unknown command or not implemented optional command.	S2
ERR_CMD_SYNTAX	0x21	Command syntax invalid	S2
ERR_OUT_OF_RANGE	0x22	Command syntax valid but command parameter(s) out of range.	S2
ERR_WRITE_PROTECTED	0x23	The memory location is write protected.	S2
ERR_ACCESS_DENIED	0x24	The memory location is not accessible.	S2
ERR_ACCESS_LOCKED	0x25	Access denied, Seed & Key is required	S2
ERR_PAGE_NOT_VALID	0x26	Selected page not available	S2
ERR_MODE_NOT_VALID	0x27	Selected page mode not available	S2
ERR_SEGMENT_NOT_VALID	0x28	Selected segment not valid	S2
ERR_SEQUENCE	0x29	Sequence error	S2
ERR_DAQ_CONFIG	0x2A	DAQ configuration not valid	S2

Error	Code	Description	Severity
ERR_MEMORY_OVERFLOW	0x30	Memory overflow error	S2
ERR_GENERIC	0x31	Generic error.	S2
ERR_VERIFY	0x32	The slave internal program verify routine detects an error.	S3

1.7.3.2 The Error Handling Matrix

1.7.3.2.1 Standard commands (STD)

Command	Error	Pre-Action	Action
CONNECT(NORMAL)	timeout t1	-	repeat ∞ times
CONNECT(USER_DEFINED)	timeout t6	wait t7	repeat ∞ times
DISCONNECT	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE	SYNCH wait t7 wait t7	repeat 2 times repeat ∞ times repeat ∞ times
GET_STATUS	timeout t1	SYNCH	repeat 2 times
SYNCH	timeout t1 ERR_CMD_SYNCH ERR_CMD_UNKNOWN	SYNCH - -	repeat 2 times – restart session

Command	Error	Pre-Action	Action
GET_COMM_MODE_INFO	timeout t1 ERR_CMD_BUSY ERR_CMD_SYNTAX	SYNCH wait t7 -	repeat 2 times repeat ∞ times retry other syntax
GET_ID	timeout t1 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE	SYNCH wait t7 - - -	repeat 2 times repeat ∞ times display error retry other syntax retry other parameter
SET_REQUEST	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE	SYNCH wait t7 wait t7 - - -	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax retry other parameter
GET_SEED	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE	SYNCH wait t7 wait t7 - - -	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax retry other parameter
UNLOCK	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_ACCESS_LOCKED ERR_SEQUENCE	SYNCH wait t7 wait t7 - - - - GET_SEED	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax retry other parameter restart session repeat 2 times

Command	Error	Pre-Action	Action
SET_MTA	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
UPLOAD	timeout t1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
SHORT_UPLOAD	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use alternative
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
BUILD_CHECKSUM	timeout t2	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times

Command	Error	Pre-Action	Action
TRANSPORT_LAYER_CMD	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
USER_CMD	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter

1.7.3.2.2 Calibration commands (CAL)

Command	Error	Pre-Action	Action
DOWNLOAD	timeout t1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_WRITE_PROTECTED	-	display error
	ERR_MEMORY_OVERFLOW	-	display error

Command	Error	Pre-Action	Action
DOWNLOAD_NEXT	timeout t1	SYNCH + DOWNLOAD	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	SET_MTA	use alternative
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED		display error
	ERR_ACCESS_LOCKED	unlock slave	repeat 2 times
	ERR_WRITE_PROTECTED	-	display error
	ERR_MEMORY_OVERFLOW		display error
	ERR_SEQUENCE	SET_MTA	repeat 2 times
DOWNLOAD_MAX	timeout t1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	SET_MTA	use alternative
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_WRITE_PROTECTED	-	display error
	ERR_MEMORY_OVERFLOW	-	display error
SHORT_DOWNLOAD	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use alternative
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_WRITE_PROTECTED	-	display error
	ERR_MEMORY_OVERFLOW	-	display error
MODIFY_BITS	timeout t1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	UPLOAD + DOWNLOAD	use alternative
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_WRITE_PROTECTED	-	display error
	ERR_MEMORY_OVERFLOW	-	display error

1.7.3.2.3 Page switching commands (PAG)

Command	Error	Pre-Action	Action
SET_CAL_PAGE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
GET_CAL_PAGE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter

Command	Error	Pre-Action	Action
GET_PAG_PROCESSOR_INFO	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAP2
	ERR_CMD_SYNTAX	-	retry other syntax
GET_SEGMENT_INFO	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAP2
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
GET_PAGE_INFO	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAP2
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
SET_SEGMENT_MODE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
GET_SEGMENT_MODE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
COPY_CAL_PAGE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter

1.7.3.2.4 Data Acquisition and Stimulation commands (DAQ)

Command	Error	Pre-Action	Action
CLEAR_DAQ_LIST	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	unlock slave	repeat 2 times
SET_DAQ_PTR	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_DAQ_ACTIVE	-	repeat 2 times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
WRITE_DAQ	timeout t1	SYNCH + SET_DAQ_PTR	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error
	ERR_ACCESS_LOCKED	unlock slave	repeat 2 times
	ERR_WRITE_PROTECTED	-	display error
	ERR_DAQ_CONFIG	-	display error
SET_DAQ_LIST_MODE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_MODE_NOT_VALID	-	retry other parameter
GET_DAQ_LIST_MODE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
START_STOP_DAQ_LIST	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_DAQ_CONFIG	-	display error
START_STOP_SYNCH	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_DAQ_CONFIG	-	display error

Command	Error	Pre-Action	Action
GET_DAQ_CLOCK	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX	SYNCH wait t7 wait t7 - -	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax
READ_DAQ	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX	SYNCH wait t7 wait t7 - -	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax
GET_DAQ_PROCESSOR_INFO	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX	SYNCH wait t7 wait t7 - -	repeat 2 times repeat ∞ times repeat ∞ times use ASAP2 retry other syntax
GET_DAQ_RESOLUTION_INFO	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX	SYNCH wait t7 wait t7 - -	repeat 2 times repeat ∞ times repeat ∞ times use ASAP2 retry other syntax
GET_DAQ_LIST_INFO	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE	SYNCH wait t7 wait t7 - - -	repeat 2 times repeat ∞ times repeat ∞ times use ASAP2 retry other syntax retry other parameter
GET_DAQ_EVENT_INFO	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE	SYNCH wait t7 wait t7 - - -	repeat 2 times repeat ∞ times repeat ∞ times use ASAP2 retry other syntax retry other parameter

Command	Error	Pre-Action	Action
FREE_DAQ	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX	SYNCH wait t7 wait t7 - -	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax
ALLOC_DAQ	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_SEQUENCE ERR_MEMORY_OVERFLOW	SYNCH wait t7 wait t7 - - - reinit DAQ reinit DAQ	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax retry other parameter repeat 2 times retry other parameter
ALLOC_ODT	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_SEQUENCE ERR_MEMORY_OVERFLOW	SYNCH wait t7 wait t7 - - - reinit DAQ reinit DAQ	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax retry other parameter repeat 2 times retry other parameter
ALLOC_ODT_ENTRY	timeout t1 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_SEQUENCE ERR_MEMORY_OVERFLOW	SYNCH wait t7 wait t7 - - - reinit DAQ reinit DAQ	repeat 2 times repeat ∞ times repeat ∞ times display error retry other syntax retry other parameter repeat 2 times retry other parameter

1.7.3.2.5 Non-volatile memory programming commands (PGM)

Command	Error	Pre-Action	Action
PROGRAM_START	timeout t3 ERR_CMD_BUSY ERR_DAQ_ACTIVE ERR_CMD_SYNTAX ERR_ACCESS_LOCKED ERR_GENERIC	SYNCH wait t7 START_STOP_x – unlock slave -	repeat 2 times repeat ∞ times repeat 2 times retry other syntax repeat 2 times restart session
PROGRAM_CLEAR	timeout t4 ERR_CMD_BUSY ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_ACCESS_DENIED ERR_ACCESS_LOCKED ERR_SEQUENCE	SYNCH+SET_MTA wait t7 - - - unlock slave -	repeat 2 times repeat ∞ times retry other syntax retry other parameter display error repeat 2 times repeat 2 times
PROGRAM	timeout t5 ERR_CMD_BUSY ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_ACCESS_DENIED ERR_ACCESS_LOCKED ERR_SEQUENCE ERR_MEMORY_OVERFLOW	SYNCH+SET_MTA wait t7 - - - unlock slave - -	repeat 2 times repeat ∞ times retry other syntax retry other parameter display error repeat 2 times repeat 2 times display error
PROGRAM_RESET	timeout t5 ERR_CMD_BUSY ERR_PGM_ACTIVE ERR_CMD_SYNTAX ERR_SEQUENCE	SYNCH wait t7 - - -	repeat 2 times repeat ∞ times repeat 2 times retry other syntax repeat 2 times

Command	Error	Pre-Action	Action
GET_PGM_PROCESSOR_INFO	timeout t1 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX	SYNCH wait t7 - -	repeat 2 times repeat ∞ times use ASAP2 retry other syntax
GET_SECTOR_INFO	timeout t1 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_MODE_NOT_VALID ERR_SEGMENT_NOT_VALID	SYNCH wait t7 - - - -	repeat 2 times repeat ∞ times use ASAP2 retry other syntax retry other parameter retry other parameter
PROGRAM_PREPARE	timeout t3 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_SEQUENCE ERR_GENERIC	SYNCH+SET_MTA wait t7 - - - - -	repeat 2 times repeat ∞ times display error retry other syntax retry other parameter repeat 2 times restart session
PROGRAM_FORMAT	timeout t1 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_SEQUENCE	SYNCH wait t7 - - - -	repeat 2 times repeat ∞ times display error retry other syntax retry other parameter repeat 2 times
PROGRAM_NEXT	timeout t1 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_ACCESS_DENIED ERR_ACCESS_LOCKED ERR_MEMORY_OVERFLOW ERR_SEQUENCE	SYNCH+PROGRAM wait t7 - - - unlock slave - -	repeat 2 times repeat ∞ times use alternative retry other syntax retry other parameter display error repeat 2 times display error repeat 2 times
PROGRAM_MAX	timeout t5 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_SEQUENCE ERR_MEMORY_OVERFLOW	SYNCH+SET_MTA wait t7 - - -	repeat 2 times repeat ∞ times use alternative repeat 2 times display error

Command	Error	Pre-Action	Action
PROGRAM_VERIFY	timeout t3 ERR_CMD_BUSY ERR_CMD_UNKNOWN ERR_CMD_SYNTAX ERR_OUT_OF_RANGE ERR_SEQUENCE ERR_GENERIC ERR_VERIFY	SYNCH wait t7 - - - - -	repeat 2 times repeat ∞ times display error retry other syntax retry other parameter repeat 2 times restart session new flashware version necessary

2 Interface to ASAM MCD 2MC description file

The following chapter describes the parameters that are independent from the Transport Layer used.

For referencing to them in the higher level *.AML (ref. Part 4 “Interface to ASAM MCD 2MC Description File”), they are grouped under the tag “Common_Parameters” in a file **XCP_common_vX_Y.aml**. (vX_Y being the current *X*CP Protocol Layer Version Number).

2.1 ASAM MCD 2MC AML for *X*CP (Common_Parameters)

```
/*************************************************************************************/
/*                                                                                   */
/*   ASAP2 meta language for XCP protocol layer V1.0                                 */
/*                                                                                   */
/*   2003-03-03                                                                      */
/*                                                                                   */
/*   Vector Informatik, Schuermans                                                   */
/*                                                                                   */
/*   Datatypes:                                                                      */
/*                                                                                   */
/*   A2ML      ASAP2          Windows     description                                */
/*   ------------------------------------------------------------------------------*/
/*   uchar     UBYTE          BYTE        unsigned 8 Bit                             */
/*   char      SBYTE          char        signed 8 Bit                               */
/*   uint      UWORD          WORD        unsigned integer 16 Bit                    */
/*   int       SWORD          int         signed integer 16 Bit                      */
/*   ulong     ULONG          DWORD       unsigned integer 32 Bit                    */
/*   long      SLONG          LONG        signed integer 32 Bit                      */
/*   float     FLOAT32_IEEE                float 32 Bit                               */
/*                                                                                   */
/*************************************************************************************/
/********************** start of PROTOCOL_LAYER ***************************/

struct Protocol_Layer {     /* At MODULE */

  uint;                      /* XCP protocol layer version */
                             /* e.g. "1.0" = 0x0100        */

  uint;                     /* T1 [ms] */
  uint;                     /* T2 [ms] */
  uint;                     /* T3 [ms] */
  uint;                     /* T4 [ms] */
  uint;                     /* T5 [ms] */
  uint;                     /* T6 [ms] */
  uint;                     /* T7 [ms] */

  uchar;                      /* MAX_CTO */
  uint;                       /* MAX_DTO */

  enum {                       /* BYTE_ORDER */
    "BYTE_ORDER_MSB_LAST"  = 0,
    "BYTE_ORDER_MSB_FIRST" = 1
  };
```

```
enum {                          /* ADDRESS_GRANULARITY */
  "ADDRESS_GRANULARITY_BYTE"      = 1,
  "ADDRESS_GRANULARITY_WORD"    = 2,
  "ADDRESS_GRANULARITY_DWORD"  = 4
};

taggedstruct {                          /* optional              */

  ("OPTIONAL_CMD" enum {          /* XCP-Code of optional command */
                                  /* supported by the slave             */

    "GET_COMM_MODE_INFO"            = 0xFB,
    "GET_ID"                        = 0xFA,
    "SET_REQUEST"                   = 0xF9,
    "GET_SEED"                      = 0xF8,
    "UNLOCK"                        = 0xF7,
    "SET_MTA"                       = 0xF6,
    "UPLOAD"                        = 0xF5,
    "SHORT_UPLOAD"                  = 0xF4,
    "BUILD_CHECKSUM"                = 0xF3,
    "TRANSPORT_LAYER_CMD"           = 0xF2,
    "USER_CMD"                      = 0xF1,
    "DOWNLOAD"                      = 0xF0,
    "DOWNLOAD_NEXT"                 = 0xEF,
    "DOWNLOAD_MAX"                  = 0xEE,
    "SHORT_DOWNLOAD"                = 0xED,
    "MODIFY_BITS"                   = 0xEC,
    "SET_CAL_PAGE"                  = 0xEB,
    "GET_CAL_PAGE"                  = 0xEA,
    "GET_PAG_PROCESSOR_INFO"    = 0xE9,
    "GET_SEGMENT_INFO"              = 0xE8,
    "GET_PAGE_INFO"                 = 0xE7,
    "SET_SEGMENT_MODE"              = 0xE6,
    "GET_SEGMENT_MODE"              = 0xE5,
    "COPY_CAL_PAGE"                 = 0xE4,
    "CLEAR_DAQ_LIST"                = 0xE3,
    "SET_DAQ_PTR"                   = 0xE2,
    "WRITE_DAQ"                     = 0xE1,
    "SET_DAQ_LIST_MODE"             = 0xE0,
    "GET_DAQ_LIST_MODE"             = 0xDF,
    "START_STOP_DAQ_LIST"           = 0xDE,
    "START_STOP_SYNCH"              = 0xDD,
    "GET_DAQ_CLOCK"                 = 0xDC,
    "READ_DAQ"                      = 0xDB,
    "GET_DAQ_PROCESSOR_INFO"    = 0xDA,
    "GET_DAQ_RESOLUTION_INFO"   = 0xD9,
    "GET_DAQ_LIST_INFO"             = 0xD8,
    "GET_DAQ_EVENT_INFO"            = 0xD7,
    "FREE_DAQ"                      = 0xD6,
    "ALLOC_DAQ"                     = 0xD5,
    "ALLOC_ODT"                     = 0xD4,
    "ALLOC_ODT_ENTRY"               = 0xD3,
    "PROGRAM_START"                 = 0xD2,
    "PROGRAM_CLEAR"                 = 0xD1,
    "PROGRAM"                       = 0xD0,
    "PROGRAM_RESET"                 = 0xCF,
    "GET_PGM_PROCESSOR_INFO"    = 0xCE,
    "GET_SECTOR_INFO"               = 0xCD,
    "PROGRAM_PREPARE"               = 0xCC,
```

```
    "PROGRAM_FORMAT"                = 0xCB,
    "PROGRAM_NEXT"                  = 0xCA,
    "PROGRAM_MAX"                   = 0xC9,
    "PROGRAM_VERIFY"                = 0xC8

  })*;

  “COMMUNICATION_MODE_SUPPORTED” taggedunion { /* optional modes supported */

    “ BLOCK” taggedstruct {

                "SLAVE";                /* Slave Block Mode supported    */
                "MASTER" struct {       /* Master Block Mode supported */

                      uchar;  /* MAX_BS */
                      uchar;  /* MIN_ST */

                    };
    };

    "INTERLEAVED" uchar;    /* QUEUE_SIZE */
   };

  "SEED_AND_KEY_EXTERNAL_FUNCTION" char[256];  /* Name of the Seed&Key function */
                                               /* including file extension      */
                                               /* without path                  */
 };

}; /********************* end of PROTOCOL_LAYER *****************************/

/***************************** start of DAQ *********************************/
struct Daq {                    /* DAQ supported, at MODULE*/

 enum {                         /* DAQ_CONFIG_TYPE */
  "STATIC"    = 0,
  "DYNAMIC" = 1
 };

 uint;                          /* MAX_DAQ */
 uint;                          /* MAX_EVENT_CHANNEL */
 uchar;                         /* MIN_DAQ */

 enum {                         /* OPTIMISATION_TYPE */
  "OPTIMISATION_TYPE_DEFAULT"                       = 0,
  "OPTIMISATION_TYPE_ODT_TYPE_16"                   = 1,
  "OPTIMISATION_TYPE_ODT_TYPE_32"                   = 2,
  "OPTIMISATION_TYPE_ODT_TYPE_64"                   = 3,
  "OPTIMISATION_TYPE_ODT_TYPE_ALIGNMENT" = 4,
  "OPTIMISATION_TYPE_MAX_ENTRY_SIZE"        = 5
 };

 enum {                         /* ADDRESS_EXTENSION */
  "ADDRESS_EXTENSION_FREE" = 0,
  "ADDRESS_EXTENSION_ODT"   = 1,
  "ADDRESS_EXTENSION_DAQ"   = 3
 };
```

```
enum {                      /* IDENTIFICATION_FIELD */
  "IDENTIFICATION_FIELD_TYPE_ABSOLUTE"                          = 0,
  "IDENTIFICATION_FIELD_TYPE_RELATIVE_BYTE"                      = 1,
  "IDENTIFICATION_FIELD_TYPE_RELATIVE_WORD"                      = 2,
  "IDENTIFICATION_FIELD_TYPE_RELATIVE_WORD_ALIGNED" = 3
};


enum {                    /* GRANULARITY_ODT_ENTRY_SIZE_DAQ */
  "GRANULARITY_ODT_ENTRY_SIZE_DAQ_BYTE"      = 1,
  "GRANULARITY_ODT_ENTRY_SIZE_DAQ_WORD"    = 2,
  "GRANULARITY_ODT_ENTRY_SIZE_DAQ_DWORD"  = 4,
  "GRANULARITY_ODT_ENTRY_SIZE_DAQ_DLONG"   = 8
};

uchar;                        /* MAX_ODT_ENTRY_SIZE_DAQ */

enum {                         /* OVERLOAD_INDICATION */
  "NO_OVERLOAD_INDICATION"       = 0,
  "OVERLOAD_INDICATION_PID"      = 1,
  "OVERLOAD_INDICATION_EVENT" = 2
};

taggedstruct {                  /* optional */
  "PRESCALER_SUPPORTED";
  "RESUME_SUPPORTED";

  block "STIM" struct {           /* STIM supported */

    enum {                  /* GRANULARITY_ODT_ENTRY_SIZE_STIM */
      "GRANULARITY_ODT_ENTRY_SIZE_STIM_BYTE"     = 1,
      "GRANULARITY_ODT_ENTRY_SIZE_STIM_WORD"   = 2,
      "GRANULARITY_ODT_ENTRY_SIZE_STIM_DWORD" = 4,
      "GRANULARITY_ODT_ENTRY_SIZE_STIM_DLONG"  = 8
    };

    uchar;                      /* MAX_ODT_ENTRY_SIZE_STIM */

    taggedstruct {              /* bitwise stimulation */
      "BIT_STIM_SUPPORTED";
    };

  };

  block "TIMESTAMP_SUPPORTED" struct {

    uint;                       /* TIMESTAMP_TICKS */

    enum { /* TIMESTAMP_SIZE */
      "NO_TIME_STAMP" = 0,
      "SIZE_BYTE"           = 1,
      "SIZE_WORD"         = 2,
      "SIZE_DWORD"        = 4
    };
```

```
  enum { /* RESOLUTION OF TIMESTAMP */
    "UNIT_1NS"      = 0,
    "UNIT_10NS"     = 1,
    "UNIT_100NS"    = 2,
    "UNIT_1US"      = 3,
    "UNIT_10US"     = 4,
    "UNIT_100US"    = 5,
    "UNIT_1MS"      = 6,
    "UNIT_10MS"     = 7,
    "UNIT_100MS"    = 8,
    "UNIT_1S"       = 9
  };

  taggedstruct {
    "TIMESTAMP_FIXED";
  };

};

"PID_OFF_SUPPORTED";

/************************ start of DAQ_LIST *************************/

(block "DAQ_LIST" struct {          /* DAQ_LIST                    */
                                    /* multiple possible           */
  uint;                             /* DAQ_LIST_NUMBER             */

  taggedstruct {                    /* optional */

    "DAQ_LIST_TYPE" enum {
      "DAQ"          = 1,           /* DIRECTION = DAQ only        */
      "STIM"         = 2,           /* DIRECTION = STIM only       */
      "DAQ_STIM" = 3                /* both directions possible    */
                                    /* but not simultaneously      */
    };

    "MAX_ODT"               uchar;  /* MAX_ODT                     */
    "MAX_ODT_ENTRIES" uchar;        /* MAX_ODT_ENTRIES */

    "FIRST_PID" uchar;              /* FIRST_PID for this DAQ_LIST */
    "EVENT_FIXED" uint;             /* this DAQ_LIST always        */
                                    /* in this event               */
    block "PREDEFINED" taggedstruct { /* predefined                */
                                      /* not configurable DAQ_LIST */
      (block “ODT” struct {
            uchar;                  /* ODT number */
            taggedstruct {
              (“ODT_ENTRY” struct {
                    uchar;  /* ODT_ENTRY number          */
                    ulong;  /* address of element        */
                    uchar;  /* address extension of element */
                    uchar;  /* size of element [AG]      */
                    uchar;  /* BIT_OFFSET                */
                })*;
            }; /* end of ODT_ENTRY */
        })*; /* end of ODT */
    }; /* end of PREDEFINED */
  };
})*;/********************* end of DAQ_LIST ***************************/
```

```
   /************************* start of EVENT *****************************/

   (block "EVENT" struct {           /* EVENT              */
                                     /* multiple possible */
     char[101];                   /* EVENT_CHANNEL_NAME             */
     char[9];                     /* EVENT_CHANNEL_SHORT_NAME */
     uint;                        /* EVENT_CHANNEL_NUMBER          */

     enum {
       "DAQ"          = 1,          /* only DAQ_LISTs              */
                                    /* with DIRECTION = DAQ   */
       "STIM"         = 2,          /* only DAQ_LISTs              */
                                    /* with DIRECTION = STIM  */
       "DAQ_STIM" = 3               /* both kind of DAQ_LISTs  */
     };

     uchar;                       /* MAX_DAQ_LIST */
     uchar;                       /* TIME_CYCLE     */
     uchar;                       /* TIME_UNIT        */
     uchar;                       /* PRIORITY         */

   })*;/******************** end of EVENT *******************************/

  }; /*end of optional at DAQ */

}; /************************* end of DAQ ************************************/


/***************************** start of DAQ_EVENT *****************************/

taggedunion Daq_Event {        /* at MEASUREMENT */

   “FIXED_EVENT_LIST” taggedstruct {
                        (“EVENT” uint)* ;
                      };
   “VARIABLE” taggedstruct {
        block “AVAILABLE_EVENT_LIST” taggedstruct {
                                    (“EVENT” uint)*;
                      };
       block “DEFAULT_EVENT_LIST” taggedstruct {
                                    (“EVENT” uint)*;
                      };
   };
}; /************************* end of DAQ_EVENT *******************************/


/***************************** start of PAG ***********************************/

struct Pag {                /* PAG supported, at MODULE */

  uchar;                        /* MAX_SEGMENTS */

  taggedstruct {                  /* optional */
    "FREEZE_SUPPORTED";
  };

}; /************************* end of PAG ************************************/
```

```
/*************************** start of PGM ***********************************/

struct Pgm {                /* PGM supported, at MODULE */

  enum {
    "PGM_MODE_ABSOLUTE"                          = 1,
    "PGM_MODE_FUNCTIONAL"                        = 2,
    "PGM_MODE_ABSOLUTE_AND_FUNCTIONAL" = 3
  };

  uchar;                        /* MAX_SECTORS */

  uchar;                        /* MAX_CTO_PGM */

  taggedstruct {                    /* optional */

    (block "SECTOR" struct {          /* SECTOR                */
                                      /* multiple possible     */
      char[101];                      /* SECTOR_NAME       */
      uchar;                          /* SECTOR_NUMBER  */

      ulong;                          /* Address                */
      ulong;                           /* Length                */

      uchar;                          /* CLEAR_SEQUENCE_NUMBER          */
      uchar;                          /* PROGRAM_SEQUENCE_NUMBER     */

      uchar;                          /* PROGRAM_METHOD     */

    })*; /* end of SECTOR */


    “COMMUNICATION_MODE_SUPPORTED” taggedunion { /* optional modes supported */

       “ BLOCK” taggedstruct {

                    "SLAVE";                  /* Slave Block Mode supported    */
                    "MASTER" struct {         /* Master Block Mode supported */

                          uchar;  /* MAX_BS_PGM */
                          uchar;  /* MIN_ST_PGM */

                        };
       };

      "INTERLEAVED" uchar;    /* QUEUE_SIZE_PGM */
     };

  };

}; /************************** end of PGM ************************************/
```

```
/***************************** start of SEGMENT *******************************/

struct Segment {              /* at MEMORY_SEGMENT */

  uchar;                      /* SEGMENT_NUMBER     */
  uchar;                      /* number of pages            */
  uchar;                      /* ADDRESS_EXTENSION */

  uchar;                      /* COMPRESSION_METHOD    */
  uchar;                      /* ENCRYPTION_METHOD     */

  taggedstruct {                  /* optional         */

    block "CHECKSUM" struct {

      enum {                      /* checksum type */
        "XCP_ADD_11"              =     1,
        "XCP_ADD_12"              =     2,
        "XCP_ADD_14"              =     3,
        "XCP_ADD_22"              =     4,
        "XCP_ADD_24"              =     5,
        "XCP_ADD_44"              =     6,
        "XCP_CRC_16"             =     7,
        "XCP_CRC_16_CITT"     =     8,
        "XCP_CRC_32"              =     9,
        "XCP_USER_DEFINED" = 255
      };

      taggedstruct {
        "MAX_BLOCK_SIZE"            ulong ;   /* maximum block size                */
                                              /* for checksum calculation          */
        "EXTERNAL_FUNCTION" char[256];  /* Name of the Checksum function */
                                              /* including file extension          */
                                              /* without path                      */
      };

    };

    (block "PAGE" struct {            /* PAGES for this SEGMENT */
                                      /* multiple possible              */
      uchar;                          /* PAGE_NUMBER                */

      enum {          /* ECU_ACCESS_TYPE */
         "ECU_ACCESS_NOT_ALLOWED"           = 0,
         "ECU_ACCESS_ WITHOUT_XCP_ONLY" =1,
         "ECU_ACCESS_ WITH_XCP_ONLY"       = 2,
         "ECU_ACCESS_ DONT_CARE"              = 3
       };

      enum {      /* XCP_READ_ACCESS_TYPE */
         "XCP_READ_ACCESS_NOT_ALLOWED"          = 0,
         "XCP_READ_ACCESS_ WITHOUT_ECU_ONLY" = 1,
         "XCP_READ_ACCESS_ WITH_ECU_ONLY"       = 2,
         “XCP_READ_ACCESS_ DONT_CARE"              = 3
       };
```

```
    enum {     /* XCP_WRITE_ACCESS_TYPE */
       "XCP_WRITE_ACCESS_NOT_ALLOWED"           = 0,
       "XCP_WRITE_ACCESS_ WITHOUT_ECU_ONLY" = 1,
       "XCP_WRITE_ACCESS_ WITH_ECU_ONLY"        = 2,
       "XCP_WRITE_ACCESS_ DONT_CARE"            = 3
    };

    taggedstruct {
      "INIT_SEGMENT" uchar; /* references segment that initialises this page */
    };

  })*; /* end of PAGE */

  (block "ADDRESS_MAPPING" struct {  /* multiple possible   */
                ulong;       /* source address      */
                ulong;       /* destination address */
                ulong;       /* length              */
  })*;

  "PGM_VERIFY" ulong; /* verification value for PGM */

 }; /* end of optional */

}; /************************** end of SEGMENT *********************************/

/********************** start of Common Parameters ***************************/

taggedstruct Common_Parameters {

 block "PROTOCOL_LAYER" struct Protocol_Layer;

 block "SEGMENT" struct Segment;

 block "DAQ" struct Daq;
 block "PAG" struct Pag;
 block "PGM" struct Pgm;

 block "DAQ_EVENT" taggedunion Daq_Event;

}; /******************** end of Common Parameters *****************************/
```

ASAM e. V.
Arnikastraße 2
D - 85635 Hoehenkirchen
Germany

Tel.: (+49) 8102 / 895317
Fax.: (+49) 8102 / 895310
E-mail: info@asam.net
Internet: www.asam.net