

XCP

Version 1.1

Part 1 - Overview

Part 1 - Overview



**Association for Standardisation of
Automation and Measuring Systems**

**Dated: 31-03-2008
© ASAM e. V.**

Status of Document

Date:	31-03-2008
Authors:	Roel Schuermans, Vector Informatik GmbH Andreas Zeiser, Vector Informatik GmbH Oliver Kitt, Vector Informatik GmbH Hans-Georg Kunz, VDO Automotive AG Hendirk Amsbeck, dSPACE GmbH Bastian Kellers, dSPACE GmbH Boris Ruoff, ETAS GmbH Reiner Motz, Robert Bosch GmbH Dirk Forwick, Robert Bosch GmbH
Version:	1.1.0
Doc-ID:	
Status:	Release
Type	

Disclaimer of Warranty

Although this document was created with the utmost care it cannot be guaranteed that it is completely free of errors or inconsistencies.

ASAM e.V. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any expressed or implied warranties of merchantability or fitness for any particular purpose. Neither ASAM nor the author(s) therefore accept any liability for damages or other consequences that arise from the use of this document.

ASAM e.V. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Revision History

This revision history shows only major modifications between release versions.

Date	Author	Filename	Comments
2008-03-31	R.Schuermans		Released document

Table of contents

0	Introduction	10
0.1	The XCP Protocol Family	10
0.2	Documentation Overview	11
0.3	Definitions and Abbreviations	12
0.4	Mapping between XCP Data Types and ASAM Data Types	13
1	XCP Features	14
1.1	Synchronous Data Transfer	15
1.1.1	The Synchronous Data Transfer Model (basic)	15
1.1.1.1	General: DAQ, STIM and ODT	15
1.1.1.2	ODT entry	16
1.1.1.3	Object Description Table (ODT)	17
1.1.1.4	DAQ list	18
1.1.1.5	Event channels	19
1.1.2	The Synchronous Data Transfer Model (optional features)	20
1.1.2.1	Dynamic DAQ Configuration	20
1.1.2.2	Advanced features	23
1.1.2.2.1	DAQ configuration storing and power-up data transfer	23
1.1.2.2.1.1	DAQ configuration storing without power-up data transfer	24
1.1.2.2.1.2	DAQ configuration storing with power-up data transfer (RESUME mode)	25
1.1.2.2.2	Master-slave synchronization	27
1.1.2.2.3	DAQ list prioritization	28
1.1.2.2.4	ODT optimization	29
1.1.2.2.5	Bitwise stimulation	31
1.1.3	The Synchronous Data Transfer DIRECTION	32
1.1.3.1	Synchronous data acquisition (DAQ)	32
1.1.3.2	Synchronous data stimulation (STIM)	33
1.2	Measurement modi	34
1.2.1	Polling	34
1.2.2	Synchronous Data Transfer, DIRECTION=DAQ, Burst, Standard	35
1.2.3	Synchronous Data Transfer, DIRECTION=DAQ, Burst, Improved	36
1.2.4	Synchronous Data Transfer, DIRECTION=DAQ, Alternating	37
1.3	Bypassing (BYP)	38
1.3.1	Bypass activation	39
1.3.2	Plausibility checks	39
1.3.3	Data consistency	39
1.3.4	Failure detection	39
1.3.5	Minimum separation time	39

1.4 Online Calibration	40
1.4.1 The Online Data Calibration Model (basic)	40
1.4.1.1 General: SECTOR, SEGMENT and PAGE	40
1.4.1.2 Logical lay-out: SEGMENT	41
1.4.1.3 Accessability: PAGE	42
1.4.2 The Online Data Calibration Model (optional features)	43
1.4.2.1 Calibration Data Page Switching	43
1.4.2.2 Advanced features	44
1.4.2.2.1 Calibration Data Page Freezing	44
1.4.3 The Online Data Calibration actions	45
1.4.3.1 Addressing	45
1.4.3.2 Master - slave	46
1.4.3.3 Page – page	47
1.5 Flash programming	48
1.5.1 The Flashing Model	48
1.5.1.1 Physical lay-out: SECTOR	48
1.5.1.2 General:	49
1.5.1.3 Absolute Access Mode : access by address	50
1.5.1.4 Functional Access Mode : access by flash area	51
1.5.1.5 Checksum Control and Program Verify	53
1.5.1.6 End of Flash Session	54
<u>2 The XCP Protocol</u>	<u>55</u>
2.1 Topology	55
2.2 The XCP communication models	57
2.2.1 The Standard communication model	57
2.2.2 The Block Transfer communication model	58
2.2.3 The Interleaved communication model	59
2.3 State machine	60
2.4 Protection Handling	63
2.5 The XCP Message (Frame) Format	64
<u>3 The Limits of Performance</u>	<u>65</u>
3.1 Generic performance parameters	65
3.2 DAQ/STIM specific performance parameters	66
3.2.1 DAQ specific parameters	67
3.2.2 STIM specific parameters	67
<u>4 Versioning</u>	<u>68</u>
4.1 The XCP Protocol Layer Version Number	68
4.2 The XCP Transport Layer Version Number	69

4.3 The Compatibility Matrix**70**

Table of diagrams:

Diagram 1 : ODT List Organization	15
Diagram 2 : DAQ List Organization	18
Diagram 3 : Static and Dynamic DAQ List Configuration	20
Diagram 4 : Static and Dynamic DAQ Configuration Sequence	22
Diagram 5 : DAQ configuration storing without power-up data transfer	24
Diagram 6 : DAQ configuration storing with power-up data transfer (RESUME mode)	25
Diagram 7 : measurement mode: polling	34
Diagram 8 : measurement mode: standard burst	35
Diagram 9 : measurement mode: improved burst	36
Diagram 10 : measurement mode: alternating	37
Diagram 11 : Bypassing	38
Diagram 12 : MIN_ST_STIM	39
Diagram 13 : Calibration Data Organisation	40
Diagram 14 : address calculation	45
Diagram 15 : Absolute Access Mode	50
Diagram 16 : Functional Access Mode	51
Diagram 17 : The XCP Topology	55
Diagram 18 : Standard Communication Model	57
Diagram 19 : Master Block Transfer	58
Diagram 20 : Slave Block Transfer	58
Diagram 21 : Interleaved Communication Model	59
Diagram 22 : The XCP slave State Machine	60
Diagram 23 : Typical use of CONNECT modes USER_DEFINED and NORMAL	61
Diagram 24 : The XCP Message (Frame) format	64

0 INTRODUCTION

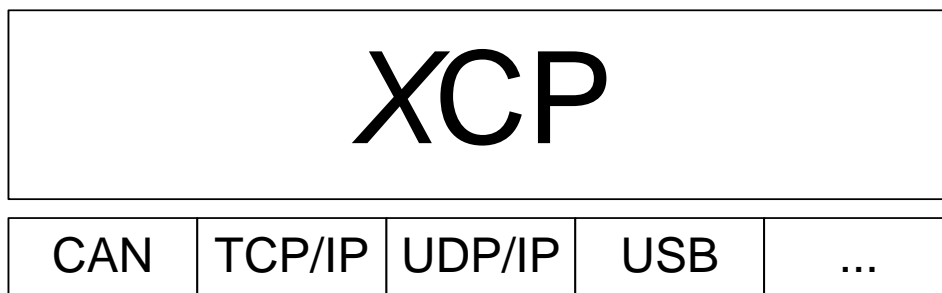
0.1 THE XCP PROTOCOL FAMILY

This document is based on experiences with the **CAN Calibration Protocol (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 XCP Specification documents describe an improved and generalized version of CCP.

The generalized protocol definition serves as standard for a protocol family and is called "XCP" (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. "XCP on CAN", "XCP on TCP/IP", "XCP on UDP/IP", "XCP on USB" and so on.



XCP is not backwards compatible to an existing CCP implementation.

0.2 DOCUMENTATION OVERVIEW

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

Part 1 “Overview” gives an overview over the XCP protocol family, the XCP features and the fundamental protocol definitions (this document).

Part 2 “Protocol Layer Specification” defines the generic protocol, which is independent from the transportation layer used.

Part 3 “Transport Layer Specification” defines the way how the XCP protocol is transported by a particular transportation layer like CAN, TCP/IP and UDP/IP.

Part 4 “Interface Specification” defines the interfaces from an XCP 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 XCP.

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 ASAM 2MC Language File
AML	ASAM 2 Meta Language
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 mand
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	E RRor Packet
EV	E Vent Packet
LEN	L ENgth
MCD	M easurement C alibration and D iagnostics
MTA	M emory T ransfer A ddress
ODT	O bject D escriptor T able
PAG	P A G ing
PGM	P ro G ra M ming
PID	P acket I Dentifier
RES	command R ESponse packet
SERV	S ERVice request packet
SPI	S erial P eripheral I nterface
STD	S Tan D ard
STIM	Data S TIMulation packet
TCP/IP	T ransfer C ontrol P rotocol / I nternet P rotocol
TS	T ime S tamp
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

0.4 MAPPING BETWEEN XCP DATA TYPES AND ASAM DATA TYPES

The following table defines the mapping between data types used in this specification and ASAM data types defined by the Project Data Harmonization Version 2.0 (ref. www.asam.net).

XCP Data Type	ASAM Data Type
BYTE	A_UINT8
WORD	A_UINT16
DWORD	A_UINT32
DLONG	A_UINT64

1 XCP FEATURES

XCP provides the following basic features:

- Synchronous data acquisition
- Synchronous data stimulation
- Online memory calibration (read / write access)
- Calibration data page initialization and switching
- Flash Programming for ECU development purposes

XCP provides the following optional, new features:

- Various transportation layers (CAN, Ethernet, USB,...)
- Block communication mode
- Interleaved communication mode
- Dynamic data transfer configuration
- Timestamped data transfer
- Synchronization of data transfer
- Prioritization of data transfer
- Atomic bit modification
- Bitwise data stimulation

XCP improves the following features compared to CCP 2.1:

- compatibility and specification
- efficiency and throughput
- power-up data transfer
- data page freezing
- auto configuration
- flash programming.

XCP was designed according to the following principles:

- Minimal Slave resource consumption (RAM, ROM, runtime)
- Efficient communication
- Simple Slave implementation

1.1 SYNCHRONOUS DATA TRANSFER

1.1.1 THE SYNCHRONOUS DATA TRANSFER MODEL (BASIC)

1.1.1.1 GENERAL: DAQ, STIM AND ODT

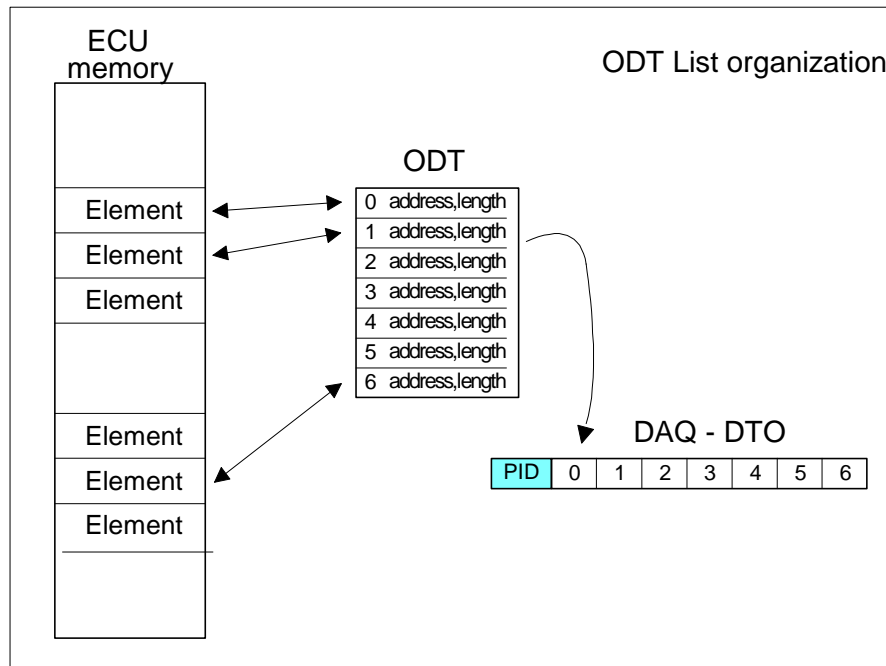


Diagram 1 : ODT List Organization

Data elements located in the slave's memory are transmitted in Data Transfer Objects DAQ from slave to master and STIM from master to slave. The Object Description Table (ODT) describes the mapping between the synchronous data transfer objects and the slave's memory.

A synchronous data transfer object is identified by its Packet Identifier (PID) that identifies the ODT that describes the contents of this synchronous data transfer object.

1.1.1.2 ODT ENTRY

An entry in an ODT references a data element by its address, the address extension, the size of the element in ADDRESS_GRANULARITY (AG) and for a data element that represents a bit, the bit offset.

GRANULARITY_ODT_ENTRY_SIZE_x determines the smallest size of a data element referenced by an ODT entry.

GRANULARITY_ODT_ENTRY_SIZE_x must not be smaller than AG.

$$\text{GRANULARITY_ODT_ENTRY_SIZE_x[BYTE]} \geq \text{AG[BYTE]}$$

Address and size of the ODT entry must meet alignment requirements regarding GRANULARITY_ODT_ENTRY_SIZE_x.

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

$$\text{Address[AG]} \bmod (\text{GRANULARITY_ODT_ENTRY_SIZE_x[BYTE]} / \text{AG[BYTE]}) = 0$$

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

$$\text{SizeOf(element described by ODT entry)[AG]} \bmod (\text{GRANULARITY_ODT_ENTRY_SIZE_x[BYTE]} / \text{AG[BYTE]}) = 0$$

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

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

The following relation must be fulfilled :

$$\text{GRANULARITY_ODT_ENTRY_SIZE_x[BYTE]} \bmod (\text{ADDRESS_GRANULARITY[BYTE]}) = 0$$

The MAX_ODT_ENTRY_SIZE_x parameters indicate the upper limits for the size of the element described by an ODT entry in ADDRESS_GRANULARITY.

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

$$\text{SizeOf(element described by ODT entry)[AG]} \leq \text{MAX_ODT_ENTRY_SIZE_x[AG]}$$

If a slave only supports elements with size = BYTE, the master has to split up multi-byte data elements into single bytes.

An ODT entry is referenced by an ODT_ENTRY_NUMBER.

1.1.1.3 OBJECT DESCRIPTION TABLE (ODT)

ODT entries are grouped in ODTs.

If DAQ lists are configured statically, MAX_ODT_ENTRIES specifies the maximum number of ODT entries in each ODT of this DAQ list.

If DAQ lists are configured dynamically, MAX_ODT_ENTRIES is not fixed and will be 0.

For every ODT the numbering of the ODT entries through ODT_ENTRY_NUMBER restarts from 0

ODT_ENTRY_NUMBER [0,1,..MAX_ODT_ENTRIES(DAQ list)-1]

1.1.1.4 DAQ LIST

Several ODTs can be grouped to a DAQ-list. XCP allows for several DAQ-lists, which may be simultaneously active. The sampling and transfer of each DAQ-list is triggered by individual events in the slave (see SET_DAQ_LIST_MODE).

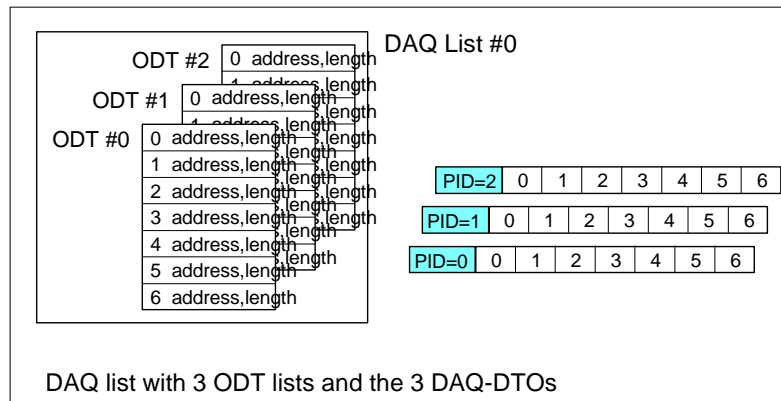


Diagram 2 : DAQ List Organization

If DAQ lists are configured statically, MAX_ODT specifies the number of ODTs for this DAQ list.

If DAQ lists are configured dynamically, MAX_ODT is not fixed and will be 0.

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].

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

ODT_NUMBER [0,1,..MAX_ODT(DAQ list)-1]

Within one and the same XCP slave device, the range for the DAQ list number starts from 0 and has to be continuous.

DAQ_LIST_NUMBER [0,1,..MIN_DAQ-1] + [MIN_DAQ,MIN_DAQ+1,..MAX_DAQ-1]

To allow reduction of the desired transfer rate, a transfer rate prescaler may be applied to the DAQ lists.(ref. PRESCALER_SUPPORTED flag in DAQ_PROPERTIES at GET_DAQ_PROCESSOR_INFO). 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 allowed for DAQ lists with DIRECTION = DAQ.

It is allowed to define “dummy” DAQ lists that contain no entries at all.

1.1.1.5 EVENT CHANNELS

XCP allows for several DAQ-lists, which may be simultaneously active. The sampling and transfer of each DAQ-list is triggered by individual events in the slave (see SET_DAQ_LIST_MODE).

An event channel builds the generic signal source that effectively determines the data transfer timing.

MAX_EVENT_CHANNEL is the number of available event channels

For each event channel 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.

XCP allows for the prioritization of 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.

The assignment of MEASUREMENT variables to event channels can optionally be controlled in the section DAQ_EVENT local at each definition of the MEASUREMENT variable. The assignment can either be fixed or variable.

If the assignment shall be fixed, a list with all event channels to be used (FIXED_EVENT_LIST) must be defined at any MEASUREMENT variable where the fixed assignment is required. The tool cannot change the assignment of the event channels for a MEASUREMENT variable with a fixed list.

If the assignment shall not be fixed but variable, a list with all valid events channels for this MEASUREMENT (AVAILABLE_EVENT_LIST) can be provided local at the MEASUREMENT. In case that such lists does not exist, all event channels provided by the ECU can be assigned by the tool.

A default assignment of the event channels to the MEASUREMENT variables can be supported by providing a list with the default event channels (DEFAULT_EVENT_LIST). This default assignment can be changed by the tool to a different assignment.

In case an AVAILABLE_EVENT_LIST is defined, the event channels in the DEFAULT_EVENT_LIST must be the same or a sub-set of the event channels in the AVAILABLE_EVENT_LIST for this MEASUREMENT variable.

Lists are possible as some MCD tools allow measurement in multiple events. Lists provide the user of a tool a simplified measurement configuration.

1.1.2 THE SYNCHRONOUS DATA TRANSFER MODEL (OPTIONAL FEATURES)

1.1.2.1 DYNAMIC DAQ CONFIGURATION

For the DAQ lists that are configurable, the slave can have certain fixed limits concerning the number of DAQ lists, the number of ODTs for each DAQ list and the number of ODT entries for each ODT.

The slave also can have the possibility to configure DAQ lists completely dynamically.

Whether the configurable DAQ lists are configurable statically or dynamically is indicated by the DAQ_CONFIG_TYPE flag in DAQ_PROPERTIES at GET_DAQ_PROCESSOR_INFO.

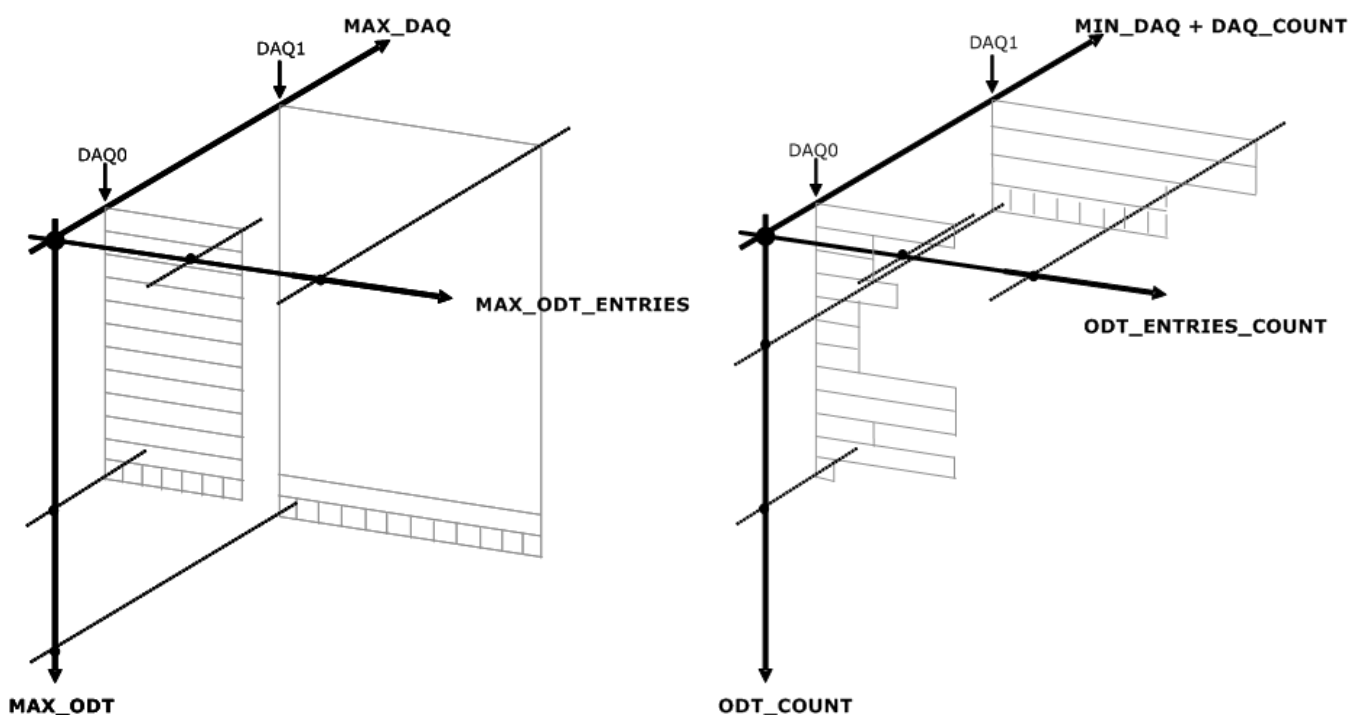


Diagram 3 : Static and Dynamic DAQ List Configuration

If DAQ lists are configured dynamically, other limits apply :

static	dynamic
MAX_DAQ	MIN_DAQ+DAQ_COUNT MAX_DAQ_ABS
MAX_ODT	MAX_ODT_DAQ_ABS MAX_ODT_STIM_ABS
MAX_ODT_ENTRIES	MAX_ODT_ENTRIES_ABS MAX_ODT_ENTRIES_DAQ_ABS MAX_ODT_ENTRIES_STIM_ABS

If DAQ lists are configured dynamically, MIN_DAQ still indicates the lower limit of the DAQ list number range.

DAQ_COUNT indicates the number of configurable DAQ lists.

For the size of an element described by an ODT entry, still the same rules concerning GRANULARITY_ODT_ENTRY_SIZE_x and MAX_ODT_ENTRY_SIZE_x have to be fulfilled.

For the allocation of FIRST_PID, still the same rules apply.

The scope of ODT_NUMBER still is local within a DAQ list.

The scope of ODT_ENTRY_NUMBER still is local within an ODT.

For the continuous numbering of DAQ list, still the same rule applies.

Dynamic DAQ list configuration is done with the commands FREE_DAQ, ALLOC_DAQ, ALLOC_ODT and ALLOC_ODT_ENTRY. These commands allow to allocate dynamically but within the above mentioned limits, a number of DAQ list, a number of ODTs to a DAQ list and a number of ODT entries to an ODT.

These commands get an ERR_MEMORY_OVERFLOW as negative response if there's not enough memory available to allocate the requested objects. If an ERR_MEMORY_OVERFLOW occurs, the complete DAQ list configuration is invalid.

During a dynamic DAQ list configuration, the master has to respect a special sequence for the use of FREE_DAQ, ALLOC_DAQ, ALLOC_ODT and ALLOC_ODT_ENTRY.

At the start of a dynamic DAQ list configuration sequence, the master always first has to send a FREE_DAQ. Secondly, with ALLOC_DAQ the master has to allocate the number of configurable DAQ lists. Then, the master has to allocate all ODTs to all DAQ lists with ALLOC_ODT commands. Finally, the master has to allocate all ODT entries to all ODTs for all DAQ lists with ALLOC_ODT_ENTRY commands.

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.


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.

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.

These rules make sure that the slave can allocate the different objects in a continuous way to the available memory which optimizes its use and simplifies its management.

 after	FREE_DAQ	ALLOC_DAQ	ALLOC_ODT	ALLOC_ODT_ENTRY
FREE_DAQ	✓	✓	ERR	ERR
ALLOC_DAQ	✓	✓	✓	ERR
ALLOC_ODT	✓	ERR	✓	✓
ALLOC_ODT_ENTRY	✓	ERR	ERR	✓

This rule implies that a new DAQ list cannot be added to an already existing configuration.

The master has to completely reconfigure the whole DAQ list configuration to include the additional DAQ list.

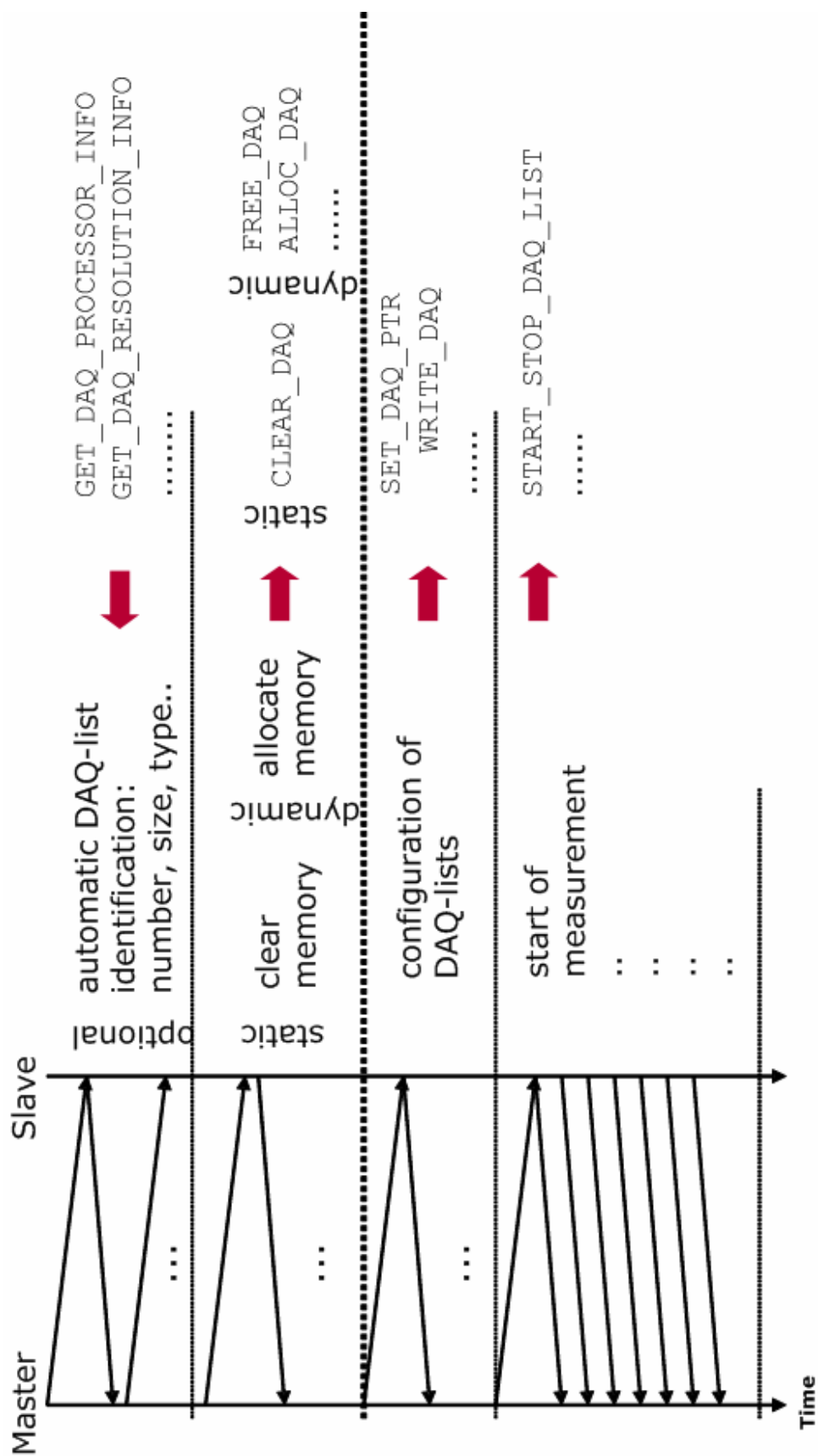


Diagram 4 : Static and Dynamic DAQ Configuration Sequence

1.1.2.2 ADVANCED FEATURES

1.1.2.2.1 DAQ configuration storing and power-up data transfer

Storing a DAQ configuration into non-volatile memory is beneficial in the following cases:

To save measurement configuration time in repetitively used, unchanged measurement configurations

To enable power-up data transfer (RESUME mode)

The XCP power-up data transfer (RESUME mode) is available since XCP version 1.0. Starting with XCP version 1.1.0 (this document), storing a DAQ configuration without automatically starting the data transfer when powering up the slave, is also possible.

With `START_STOP_DAQ_LIST(Select)`, the master can select a DAQ list to be part of a DAQ list configuration the slave stores into non-volatile memory.

The master has to calculate a Session Configuration Id based upon the current configuration of the DAQ lists selected for storing.

The master has to store this Session Configuration Id internally for further use.

The master also has to send the Session Configuration Id to the slave with `SET_REQUEST`.

If `STORE_DAQ_REQ_RESUME` or `STORE_DAQ_REQ_NO_RESUME` is set and the appropriate conditions are met, the slave then has to save all DAQ lists which have been selected, into non-volatile memory.

If `STORE_DAQ_REQ_RESUME` or `STORE_DAQ_REQ_NO_RESUME` is set, the slave also has to store the Session Configuration Id in non-volatile memory. It will be returned in the response of `GET_STATUS`.

This allows the master device to verify that automatically started DAQ lists contain the expected data transfer configuration.

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.

In principle the slave needs to take care of the status dependent on the requests and their progress and must report the status with `GET_STATUS` accordingly.

1.1.2.2.1.1 DAQ configuration storing without power-up data transfer

The purpose of DAQ configuration storing without power-up data transfer is to enable faster start of not changed DAQ configurations (DAQ/STIM).

The STORE_DAQ_SUPPORTED entry in the IF_DATA indicates that the slave can store DAQ configurations.

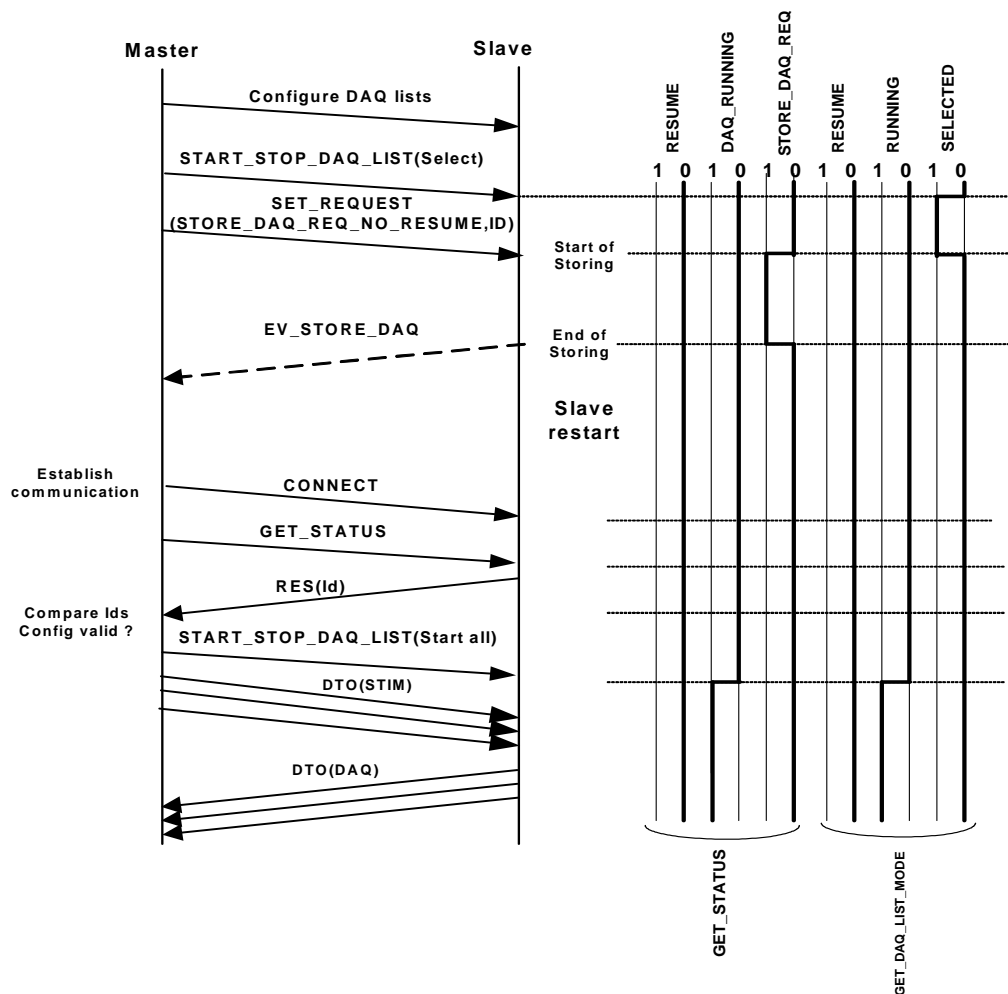


Diagram 5 : DAQ configuration storing without power-up data transfer

A configured DAQ setup can be stored via a SET_REQUEST (STORE_DAQ_REQ_NO_RESUME).

The Master can detect if a DAQ configuration was stored to the slave by checking the Session Configuration Id which is returned by GET_STATUS. If it does not equal zero a configuration is present.

1.1.2.2.1.2 DAQ configuration storing with power-up data transfer (RESUME mode)

The resume mode is one state of the state machine.

The purpose of the resume mode is to enable automatic data transfer (DAQ,STIM) directly after the power up of the slave.

The RESUME_SUPPORTED flag in DAQ_PROPERTIES at GET_DAQ_PROCESSOR_INFO indicates that the slave can be set into RESUME mode.

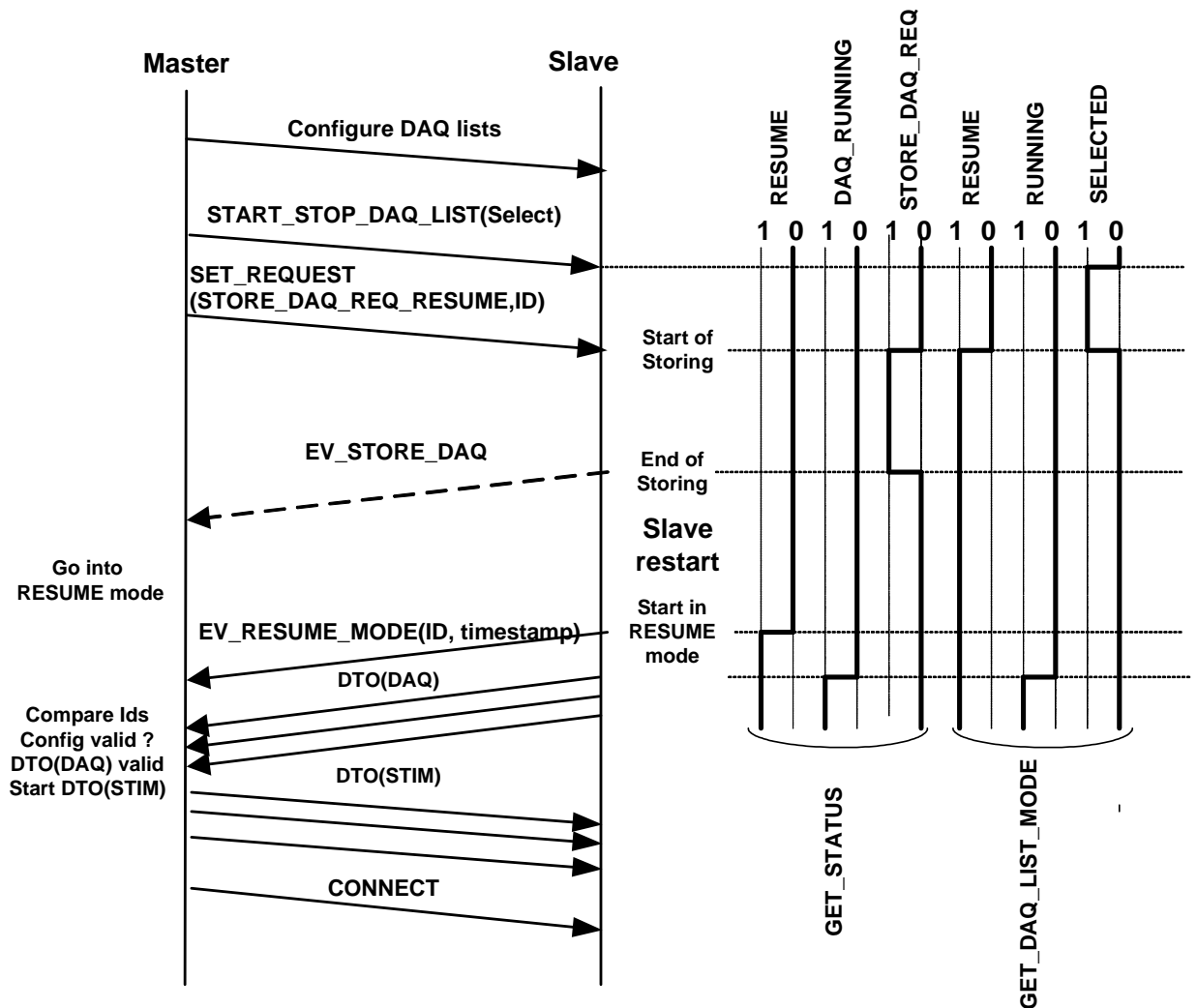


Diagram 6 : DAQ configuration storing with power-up data transfer (RESUME mode)

With GET_STATUS, the master can identify whether a slave is in RESUME mode.

With GET_DAQ_LIST_MODE the master can identify whether a DAQ list is part of a DAQ list configuration the slave uses when in RESUME mode.

If STORE_DAQ_REQ_RESUME is set, the slave internally has to set the RESUME bit of those DAQ lists that previously have been selected with START_STOP_DAQ_LIST(select).

RESUME mode is allowed for both directions, DAQ and STIM.

On each power up, the slave has to restore the DAQ lists and send an EV_RESUME_MODE to the master

Position	Type	Description
0	BYTE	Packet ID: Event 0xFD
1	BYTE	EV_RESUME_MODE: 0x00
2,3	WORD	Session Configuration Id from slave
4..7	DWORD	Current slave Timestamp (optional)

The EV_RESUME_MODE has to contain the Session Configuration Id.

If the slave has the TIMESTAMP_SUPPORTED flag set in GET_DAQ_PROCESSOR_INFO, in Current slave Timestamp the EV_RESUME_MODE also has to contain the current value of the data acquisition clock. The Current slave Timestamp has the format specified by the GET_DAQ_RESOLUTION_INFO command.

For DAQ list with DIRECTION = DAQ, then the slave automatically will start transferring DAQ packets to the master, even before any XCP command was sent by the master.

For DAQ list with DIRECTION = STIM, then the slave automatically will be ready for receiving STIM packets from the master, even before any XCP command was sent by the master.

For DAQ lists automatically started at power up, the Current Mode of GET_DAQ_LIST_MODE will be RESUME and RUNNING.

RESUME mode implies that any data transfer will only start after the physical communication channel is up and running.

The master and the slave have to remember all the necessary communication parameters that were used when a SET_REQUEST(STORE_DAQ_REQ_RESUME) was sent. At power-up, both the master and the slave have to use these same parameters for the automatic data transfer.

At power-up the slave's unlock state can be different from its unlock state at the moment that the SET_REQUEST(STORE_DAQ_REQ) was sent.

1.1.2.2.2 Master-slave synchronization

The GET_DAQ_CLOCK command provides a way to synchronize the clocks in the master and the slave device, by calculation of an offset.

1.1.2.2.3 DAQ list prioritization

XCP allows the prioritization of DAQ-lists. The limited length of the DTOs together with the prioritization mechanism, makes sure that with an acceptable delay a DAQ list with higher priority can interrupt the transfer of a DAQ list with lower priority.

1.1.2.2.4 ODT optimization

XCP allows DTO optimization on ODT level.

To support this feature the slave implementation may use one or more specific copy routines in order to make full use of the CPUs architecture for copying data. Optimization can be done in a way to minimize runtime, or to maximize the effective data transfer rate, or even both.

However these copy routines may need specific ODT structures. To get the advantage of DAQ optimization, the master should configure the ODTs in a way to fit the requirements of the copy routines.

The `Optimization_Method` property indicates the kind of optimization method, used by the slave implementation. It should be used by the master to determine the method, used for configuring the ODTs.

`Optimization_Method` is a global DAQ property, valid for all ODTs and DAQ lists. The `Optimization_Method` flags are located in `DAQ_KEY_BYTE` at `GET_DAQ_PROCESSOR_INFO`.

The following Optimization Methods are defined:

OM_DEFAULT:	No special requirements. <code>GRANULARITY_ODT_ENTRY_SIZE_DAQ</code> , <code>GRANULARITY_ODT_ENTRY_SIZE_STIM</code> , <code>MAX_ODT_ENTRY_SIZE_DAQ</code> and <code>MAX_ODT_ENTRY_SIZE_STIM</code> must be considered.
OM_ODT_TYPE_16:	Type specific copy routines are used on ODT level. WORD (16 Bit) is the largest type, supported by the copy routines. <code>GRANULARITY_ODT_ENTRY_SIZE_DAQ</code> and <code>GRANULARITY_ODT_ENTRY_SIZE_STIM</code> define the smallest type. All entries within the same ODT should be of the same type. Length and address of each ODT entry must meet the alignment requirements of the ODT type. <code>MAX_ODT_ENTRY_SIZE_DAQ</code> and <code>MAX_ODT_ENTRY_SIZE_STIM</code> must be considered.
OM_ODT_TYPE_32:	Type specific copy routines are used on ODT level. DWORD (32 Bit) is the largest type, supported by the copy routines. <code>GRANULARITY_ODT_ENTRY_SIZE_DAQ</code> and <code>GRANULARITY_ODT_ENTRY_SIZE_STIM</code> define the smallest type. All entries within the same ODT should be of the same type. Length and address of each ODT entry must meet the alignment requirements of the ODT type. <code>MAX_ODT_ENTRY_SIZE_DAQ</code> and <code>MAX_ODT_ENTRY_SIZE_STIM</code> must be considered.
OM_ODT_TYPE_64:	Type specific copy routines are used on ODT level. DLONG (64 Bit) is the largest type, supported by the copy routines. <code>GRANULARITY_ODT_ENTRY_SIZE_DAQ</code> and <code>GRANULARITY_ODT_ENTRY_SIZE_STIM</code> define the smallest type. All entries within the same ODT should be of the same type. Length and address of each ODT entry must meet the alignment requirements of the ODT type. <code>MAX_ODT_ENTRY_SIZE_DAQ</code> and <code>MAX_ODT_ENTRY_SIZE_STIM</code> must be considered.

OM_ODT_ALIGNMENT: Within one ODT all kind of data types are allowed. However they must be arranged in alignment order. Large data types first and small data types last.

Length and address of each ODT entry must meet the alignment requirements.

GRANULARITY_ODT_ENTRY_SIZE_DAQ, GRANULARITY_ODT_ENTRY_SIZE_STIM, MAX_ODT_ENTRY_SIZE_DAQ and MAX_ODT_ENTRY_SIZE_STIM must be considered.

OM_MAX_ENTRY_SIZE: Only ODT entries of a fixed length are supported (for example data blocks of 16 bytes).

The Length is defined by MAX_ODT_ENTRY_SIZE_DAQ and MAX_ODT_ENTRY_SIZE_STIM.

Length and address of each ODT entry must meet the alignment requirements determined by GRANULARITY_ODT_ENTRY_SIZE_DAQ and GRANULARITY_ODT_ENTRY_SIZE_STIM.

If the configuration of an ODT does not correspond to the requested optimization method,

the slave can answer with an ERR_DAQ_CONFIG message to show that this configuration can not be handled. The configuration of all DAQ lists is not valid.

the slave implementation can be tolerant. In this case it will handle the configuration, but in a non-optimal way.

1.1.2.2.5 Bitwise stimulation

The BIT_STIM_SUPPORTED flag in DAQ_PROPERTIES at GET_DAQ_PROCESSOR_INFO indicates that the slave supports bit wise data stimulation.

The BIT_OFFSET field at WRITE_DAQ allows the transfer 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 bytes. 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.

1.1.3 THE SYNCHRONOUS DATA TRANSFER DIRECTION

1.1.3.1 SYNCHRONOUS DATA ACQUISITION (DAQ)

By means of the DIRECTION flag, a DAQ-list can be put in Synchronous Data Acquisition mode. By means of DAQ with $0x00 \leq \text{PID} \leq 0xFB$ the slave has to transfer the contents of the elements defined in each ODT of the DAQ-list to the master.

When processing an ODT, the slave can go to the next ODT as soon as it finds an element with size = 0 in the current ODT or all ODT entries of this ODT have been processed. When processing a DAQ list, the slave can go to the next DAQ list as soon as it finds an element with size = 0 at the first ODT entry of the first ODT of this DAQ list or all ODTs of this DAQ list have been processed.

The slave has to sample the elements consistently. When a DAQ list is triggered, the slave at least has to sample the data for one and the same ODT in a consistent way, so consistency on the ODT level is always guaranteed. However, the slave may need some time to sample and transmit the complete DAQ list with all its ODTs.

When a new event cycle is triggered before the transfer of the previous cycle has been finished, the slave is said to have an "OVERLOAD situation". An overflow indication therefore is a temporary state. All sample values which were sent before the first overflow indication, are not affected.

The slave device may indicate this OVERLOAD situation to the master. The kind of OVERLOAD indication is indicated by the OVERLOAD_x flags in DAQ_PROPERTIES at GET_DAQ_PROCESSOR_INFO. The slave's reaction on an OVERLOAD situation is implementation dependent.

With CONSISTENCY DAQ at the definition of an Event Channel in the ASAM MCD 2MC description file, the slave can indicate that for this Event all data that belong to one and the same DAQ list are sampled consistently.

With CONSISTENCY EVENT at the definition of an Event Channel in the ASAM MCD 2MC description file, the slave can indicate that for this Event all data are sampled consistently.

If there's only one DAQ list associated with this Event, CONSISTENCY DAQ has the same meaning as CONSISTENCY EVENT.

If more than one DAQ-list is associated with this Event, CONSISTENCY DAQ implies that the data of every specific DAQ list in this Event are sampled consistently within the DAQ list. However there's no data consistency between data that are processed in different DAQ lists.

If more than one DAQ-list is associated with this Event, CONSISTENCY EVENT implies that all data of all DAQ lists in this Event are sampled consistently.

1.1.3.2 SYNCHRONOUS DATA STIMULATION (STIM)

Synchronous Data Stimulation is the inverse mode of Synchronous Data Acquisition. By means of the DIRECTION flag, a DAQ-list can be put in Synchronous Data Stimulation mode. Data for stimulation is transmitted in DTO packets. An ODT describes the mapping between the DTO and the slave's memory. By means of STIM with $0x00 \leq \text{PID} \leq 0xBF$ the master has to transfer the contents of the elements defined in each ODT of the DAQ-list to the slave.

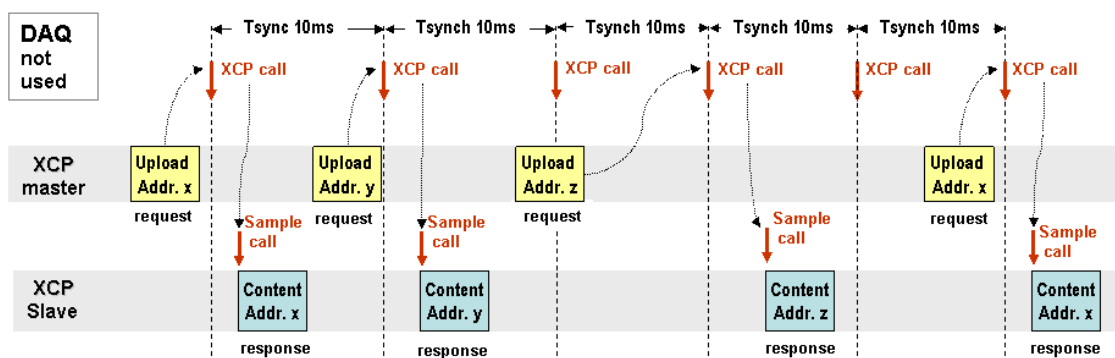
The STIM processor buffers incoming data stimulation packets. When an event occurs which triggers a DAQ list in data stimulation mode, the buffered data is transferred to the slave device's memory.

1.2 MEASUREMENT MODI

1.2.1 POLLING

Simple Measurement Mode

Polling



only example: Call of XCP inside the XCP slave every 10ms

Diagram 7 : measurement mode: polling

The easiest way for measurement uses the polling method. The characteristic of this method is that every measurement value is requested by the XCP master in principle with an extra XCP command. The effective sample rate is based on the performance of the XCP slave and of the performance of the XCP master.

An XCP timestamp mechanism cannot be used.

There's no consistency between the different measurement values.

There is no need to set up a measurement configuration (configuring DAQ lists) for the XCP slave.

The following XCP commands can be used for polled measurement:

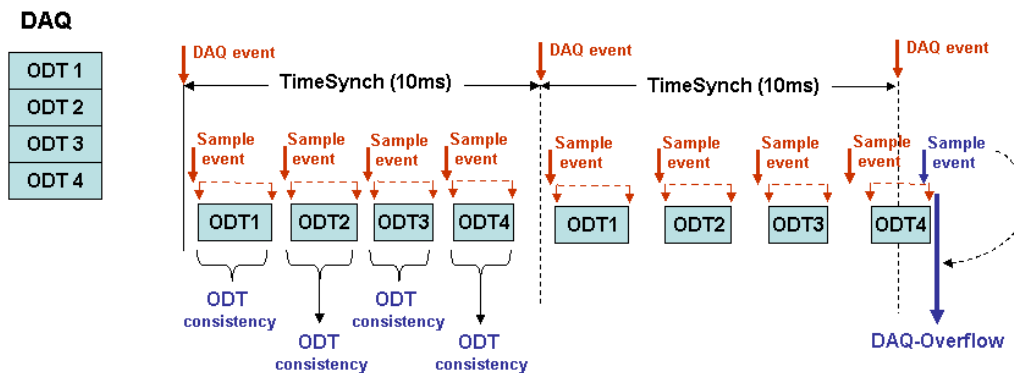
- SHORT_UPLOAD (recommended)
- or

- SET_MTA
- UPLOAD

1.2.2 SYNCHRONOUS DATA TRANSFER, DIRECTION=DAQ, BURST, STANDARD

Standard Burst Mode

DAQ event synchronous with ODT consistency



only example: DAQ recurrency 10ms

Diagram 8 : measurement mode: standard burst

The standard measurement mode of XCP uses an optimized method for reading ECU-internal values. During a configuration phase in advance the master can specify all data of interest via definition of ODTs which are assigned to a DAQ event. After starting the measurement the XCP slave will send the ODTs independently of the XCP master and only based on an internal DAQ trigger event.

The characteristic of the measurement data is ODT consistency. ODT consistency means that the complete content of an ODT is sampled at the same time. The observance of the requested sample rate is based on the performance of the XCP slave and the transmission time of a complete DAQ message.

As an optional feature the XCP timestamp mechanism can be used.

ODT consistency is a minimum requirement of XCP for measurement data.

A DAQ overflow is an event, i.e. a message generated from the XCP slave, to inform the XCP master that the measurement requirements were violated.

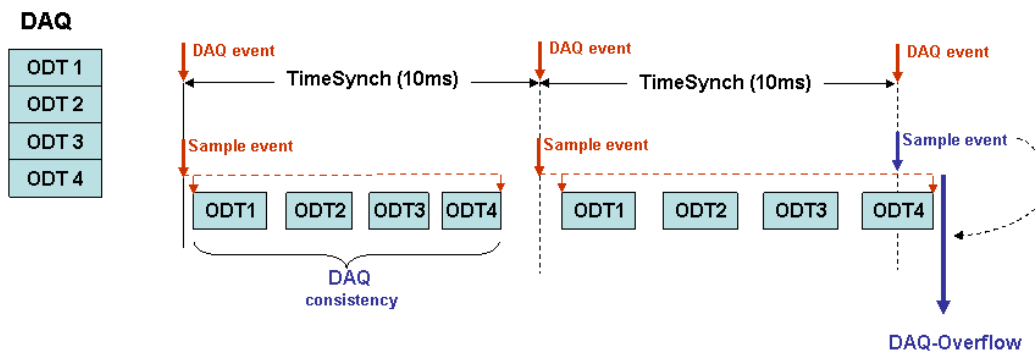
In order to configure this mode, the following command is necessary:

- SET_DAQ_LIST_MODE

1.2.3 SYNCHRONOUS DATA TRANSFER, DIRECTION=DAQ, BURST, IMPROVED

Improved Burst Mode

DAQ event synchronous with DAQ consistency



only example: DAQ recurrency 10ms

Diagram 9 : measurement mode: improved burst

The improved measurement mode is based on the standard measurement mode, but uses a single event driven method for data sampling. The characteristic of the measurement data is DAQ consistency. DAQ consistency means that all ODTs of one DAQ are sampled at the same time. The observance of the requested sample rate is based on the performance of the XCP slave and the transmission time of a complete DAQ message.

As an optional feature the XCP timestamp mechanism can be used.

A DAQ overflow is an event, i.e. a message generated from the XCP slave, to inform the XCP master that the measurement requirements were violated.

In order to configure this mode, the following command is necessary:

- SET_DAQ_LIST_MODE

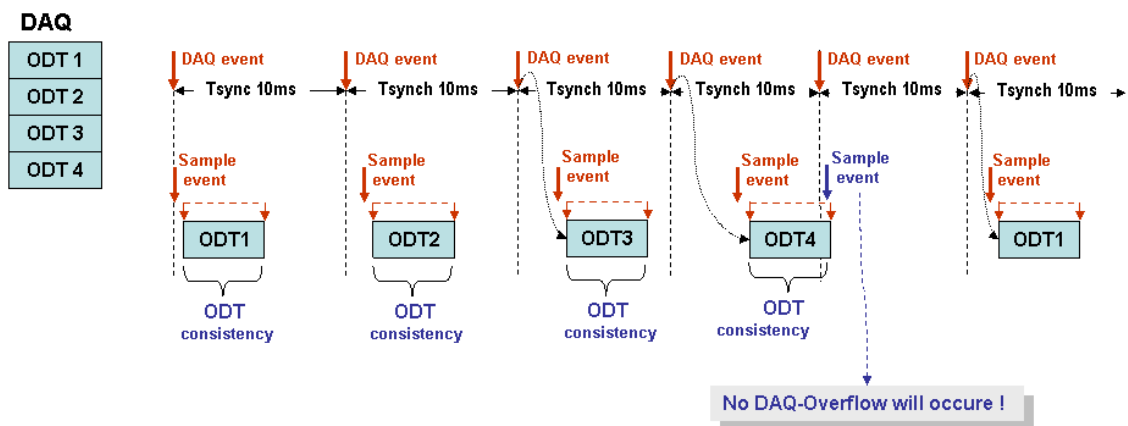
With CONSISTENCY DAQ or CONSISTENCY EVENT at the definition of an Event Channel in the ASAM MCD 2MC description file, the slave can indicate what kind of data consistency exists when data are processed within this Event.

s

1.2.4 SYNCHRONOUS DATA TRANSFER, DIRECTION=DAQ, ALTERNATING

Alternating Display Mode

DAQ event synchronous with simplified DAQ consistency



only example: DAQ recurrency (= ODT recurrency) ~ 10ms

Diagram 10 : measurement mode: alternating

XCP offers a lean measurement mode with a very low performance. Goal of this mode is only to display ECU internal data with limited consumption of ECU resources or XCP slave resources. Although all ODTs are formally assigned to one DAQ list, sample gaps are allowed and will not be reported. Therefore these data are not qualified for measurement.

The XCP mechanism for timestamps is not allowed.

There's a reuse of the configuration structure of the standard XCP measurement mode, but the alternating mode itself works differently. Every DAQ event will cause the sample of one ODT, but internal delays will not cause a DAQ overflow event. Therefore, the master does not know how long the real refresh cycle of the complete DAQ lasts. Only the ODT sequence itself is stable.

In order to configure this mode, the following command is necessary:

- SET_DAQ_LIST_MODE

The ALTERNATING flag selects the alternating display mode.

The master is not allowed to set the ALTERNATING flag and the TIMESTAMP flag at the same time. Therefore a slave in its ASAM MCD 2MCD description file is not allowed to use TIMESTAMAP_FIXED and DAQ_ALTERNATING_SUPPORTED at the same time.

The master can set the ALTERNATING flag only when setting DIRECTION = DAQ at the same time.

1.3 BYPASSING (BYP)

Bypassing can be implemented by making use of Synchronous Data Acquisition and Synchronous Data Stimulation simultaneously.

For bypassing, at least two DAQ lists are required for transferring data synchronously between the ECU and the bypassing tool, i.e. one DAQ list with direction DAQ and one DAQ list with direction STIM.

Furthermore specific event channels are required, which are intended for bypassing purposes. The keyword `COMPLEMENTARY_EVENT_CHANNEL_NUMBER` in the XCP IF_DATA section of the ASAM MCD 2MC description file can be used to make a combination of two event channels building a bypassing raster. A bypassing tool can use this information to display the available bypass rasters of an ECU.

For bypassing, two approaches are possible: bypassing without and bypassing with one or more sample steps delay.

In the first approach, typically input data of the bypass function are sent to the bypassing tool before the original ECU function is called and the respective output variables are stimulated at the end of the original function. Thus, the output data sent to the ECU are calculated based on the input data of the same sample step.

In the second approach, the output data is based on inputs of a previous sample step. Typically, at the end of an ECU task, input data for the bypass function are sent to the bypassing tool. In the following sample step the respective output variables in the ECU are stimulated after the execution of the original ECU function. This approach is usually taken when a slow transport layer is used.

State-of-the-art bypassing also requires the administration of the bypassed functions and additional implementation specific mechanisms.

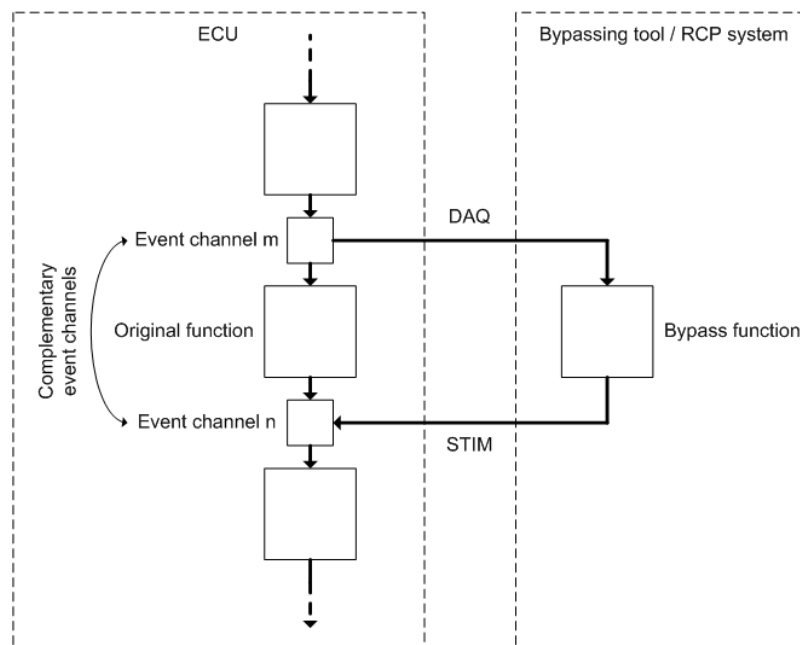


Diagram 11 : Bypassing

1.3.1 BYPASS ACTIVATION

The adoption of the ECU code to support a bypass raster, is called a bypass hook. For security reasons, a bypass hook may need to be activated before it is functional. The mechanism to enable a bypass hook is implementation specific and not part of this specification. It is recommended to activate bypass hooks by means of XCP, e.g. by calibrating a specific calibration parameter.

1.3.2 PLAUSIBILITY CHECKS

The XCP slave should perform plausibility checks on the data it receives through data stimulation. The borders (e.g. minimum and maximum values) and actions of these checks may be set by standard calibration methods. No special XCP commands are needed for this.

1.3.3 DATA CONSISTENCY

In case the stimulation data are transmitted in several ODTs, the XCP slave should buffer all incoming ODTs and stimulate the data consistently after all ODTs have been received from the bypassing tool.

When performing bypassing without a sample step delay, the XCP slave also may need to wait for a specific amount of time to receive all incoming ODTs because the data exchange with the bypassing tool and the calculation of the bypass function may take longer than the execution of the original ECU function.

If a timeout occurs in the current sample step, it is implementation specific whether the slave uses inconsistent data, old data from a previous sample step, or e.g. default data for stimulation. Furthermore, the slave may indicate this error by transmitting an EV_STIM_TIMEOUT event packet.

1.3.4 FAILURE DETECTION

For bypassing it is essential that for each sample step all stimulation data have been received. In case of transmission errors or a broken communication link, the slave should be able to recognize an instable bypassing connection and perform appropriate actions. The slave may indicate an instable bypass communication by transmitting an EV_SESSION_TERMINATED event packet.

1.3.5 MINIMUM SEPARATION TIME

The slave should be able to receive stimulation data in several ODTs from the bypassing tool. If the slave needs time from one to the next ODT, it must be indicated to the bypassing tool. The parameter MIN_ST_STIM is designed for this purpose.

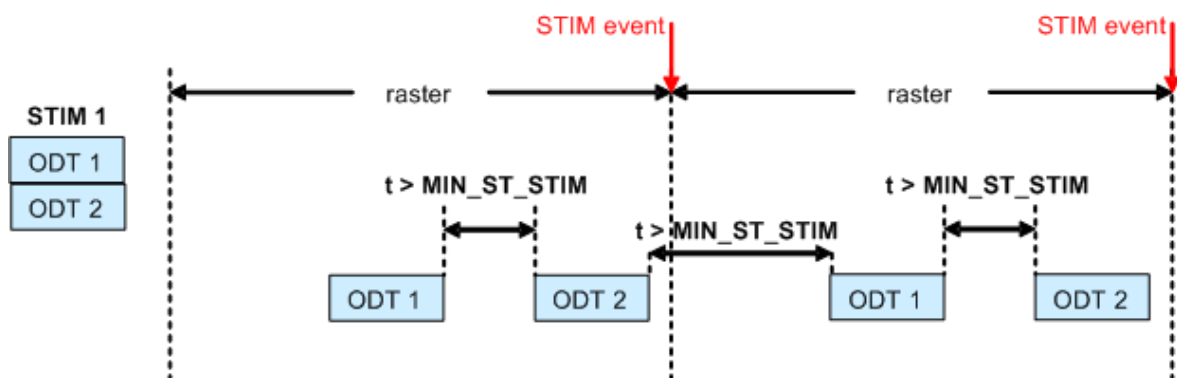


Diagram 12 : MIN_ST_STIM

1.4 ONLINE CALIBRATION

1.4.1 THE ONLINE DATA CALIBRATION MODEL (BASIC)

1.4.1.1 GENERAL: SECTOR, SEGMENT AND PAGE

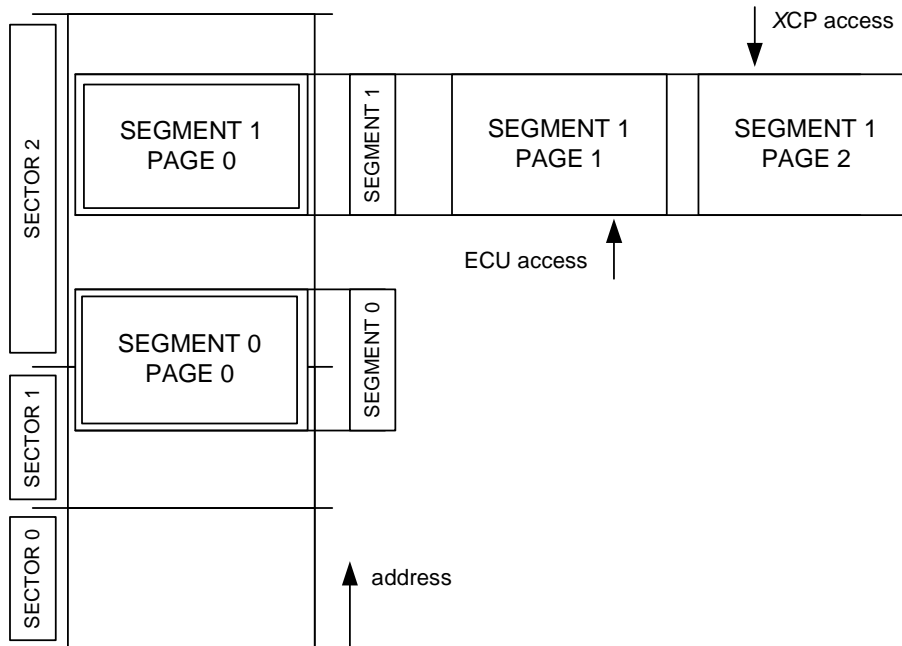


Diagram 13 : Calibration Data Organisation

The slave's memory lay-out is described as one continuous physical space. Elements are referenced with a 40 bit address (32 bit XCP address + 8 bit XCP address extension).

The physical lay-out is described with objects called SECTORS. SECTOR limits and sizes are important when reprogramming (Flashing) the slave device.

The logical lay-out is described with objects called SEGMENTS. SEGMENTS describe WHERE the calibratable data objects are located in the slave's memory. The start address and size of a SEGMENT doesn't have to respect the limitations given by the start addresses and sizes of the SECTOR lay-out.

Every SEGMENT can have multiple PAGES. The PAGES of a SEGMENT describe the same data on the same addresses, but with different properties e.g. different values or read/write access.

When searching for data to be used by the control algorithms in the slave, at any moment (for every SEGMENT) the slave can access only one PAGE. This PAGE is called the "active PAGE for ECU access for this SEGMENT".

When referencing data with XCP commands, at any moment (for every SEGMENT) the XCP master can access only one PAGE. This PAGE is called the "active PAGE for XCP access for this SEGMENT".

The active PAGE for ECU access and XCP access can be switched independently. The active PAGE can be switched independently for every SEGMENT.

1.4.1.2 LOGICAL LAY-OUT: SEGMENT

The logical lay-out of the slave's memory is described with objects called SEGMENTS. SEGMENTS describe WHERE the calibratable data objects are located in the slave's memory. The start address and size of a SEGMENT doesn't have to respect the limitations given by the start addresses and sizes of the SECTOR lay-out (ref. Flashing).

A SEGMENT is described with the normal ASAM MCD2 keyword MEMORY_SEGMENT which contains information like Name, Address, Size and Offsets for Mirrored Segments.

The XCP specific information is inside an IF_DATA section.

For having a 40 bit address space, every SEGMENT is having an address extension which is valid for all calibratable objects that are located within this SEGMENT.

XCP references a SEGMENT by its SEGMENT_NUMBER.

Within one and the same XCP slave device, the range for the SEGMENT_NUMBER starts from 0 and has to be continuous.

SEGMENT_NUMBER [0,1,..255]

1.4.1.3 ACCESSABILITY: PAGE

Every SEGMENT can have multiple PAGES.

The PAGES of a SEGMENT describe the same data on the same addresses, but with different properties e.g. different values or read/write access.

Every SEGMENT always at least has to have 1 PAGE, called PAGE 0.

The slave always has to initialize all its PAGES for all its SEGMENTS.

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

With GET_CAL_PAGE, the master can get the current active PAGES for XCP and ECU access of the slave.

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 XCP master does NOT access this PAGE at the same time, only if the XCP master accesses this page at the same time, or the ECU doesn't care whether the XCP master accesses this page at the same time or not.

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

If the XCP master can access this PAGE, the XCP_READ_ACCESS_x flags indicate whether the XCP 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 XCP master doesn't need to care whether the ECU accesses this page at the same time or not.

If the XCP master can access this PAGE, the XCP_WRITE_ACCESS_x flags indicate whether the XCP 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 XCP master doesn't need to care whether the ECU accesses this page at the same time or not.

For every SEGMENT the numbering of the PAGES through PAGE_NUMBER restarts from 0

PAGE_NUMBER(Segment j) [0,1,..255]

1.4.2 THE ONLINE DATA CALIBRATION MODEL (OPTIONAL FEATURES)

1.4.2.1 CALIBRATION DATA PAGE SWITCHING

If the slave supports the optional commands `GET_CAL_PAGE` and `SET_CAL_PAGE`, page switching is supported.

When searching for data to be used by the control algorithms in the slave, at any moment (for every SEGMENT) the slave can access only one PAGE. This PAGE is called the “active PAGE for ECU access for this SEGMENT”.

When referencing data with XCP commands, at any moment (for every SEGMENT) the XCP master can access only one PAGE. This PAGE is called the “active PAGE for XCP access for this SEGMENT”.

With `GET_CAL_PAGE`, the master can request the slave to answer the current active PAGE for ECU or XCP access for this SEGMENT.

With `SET_CAL_PAGE`, the master can set the current active PAGE for ECU or XCP access for this SEGMENT.

The master has the full control for switching the pages. The slave can not switch its pages autonomously.

The active PAGE for ECU access and XCP access can be switched independently.

The active PAGE can be switched independently for every SEGMENT.

The master also can switch all SEGMENTS synchronously to the same PAGE.

The master has to respect the constraints given by the `XCP_ACCESS_TYPE` and `ECU_ACCESS_TYPE`.

1.4.2.2 ADVANCED FEATURES

1.4.2.2.1 Calibration Data Page Freezing

The `FREEZE_SUPPORTED` flag in `PAG_PROPERTIES` at `GET_PAG_PROCESSOR_INFO` indicates that all `SEGMENTS` can be put in `FREEZE` mode.

With `SET_SEGMENT_MODE` the master can select a `SEGMENT` for freezing.

With `GET_SEGMENT_MODE` the master can identify whether a `SEGMENT` has been selected for `FREEZING`.

With `STORE_CAL_REQ` in `SET_REQUEST`, the master requests the slave to save calibration data into non-volatile memory.

For each `SEGMENT` that is in `FREEZE` mode, the slave has to save the current active `XCP PAGE` for this `SEGMENT` into `PAGE 0` of the `INIT_SEGMENT` of this `PAGE`.

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.

1.4.3 THE ONLINE DATA CALIBRATION ACTIONS

1.4.3.1 ADDRESSING

The slave's memory lay-out is described as one continuous physical space. Elements are referenced with a 40 bit address (32 bit XCP address + 8 bit XCP address extension).

The address extension is taken from the SEGMENT to which the currently referenced address belongs.

The address range at MEMORY_SEGMENT describes the addresses from which the master can generate a file that can be programmed into the slave and then will result in a normal operating slave.

For checking whether a CHARACTERISTIC belongs to a MEMORY_SEGMENT, the master has to take the address written at CHARACTERISTIC, if applicable apply the ECU_CALIBRATION_OFFSET and if applicable the dereferencing of the NearPointer and then check this resulting address to be part of the MEMORY_SEGMENT.

For the (destination) address used in SET_MTA , SHORT_UPLOAD and SHORT_DOWNLOAD, the master has to take the address as calculated above (take address at CHARACTERISTIC, apply ECU_CALIBRATION_OFFSET, dereference) and if applicable apply an ADDRESS_MAPPING from the calculated (source) address to a mapped (destination) address.

ADDRESS_MAPPING can be different for different parts of a SEGMENT.

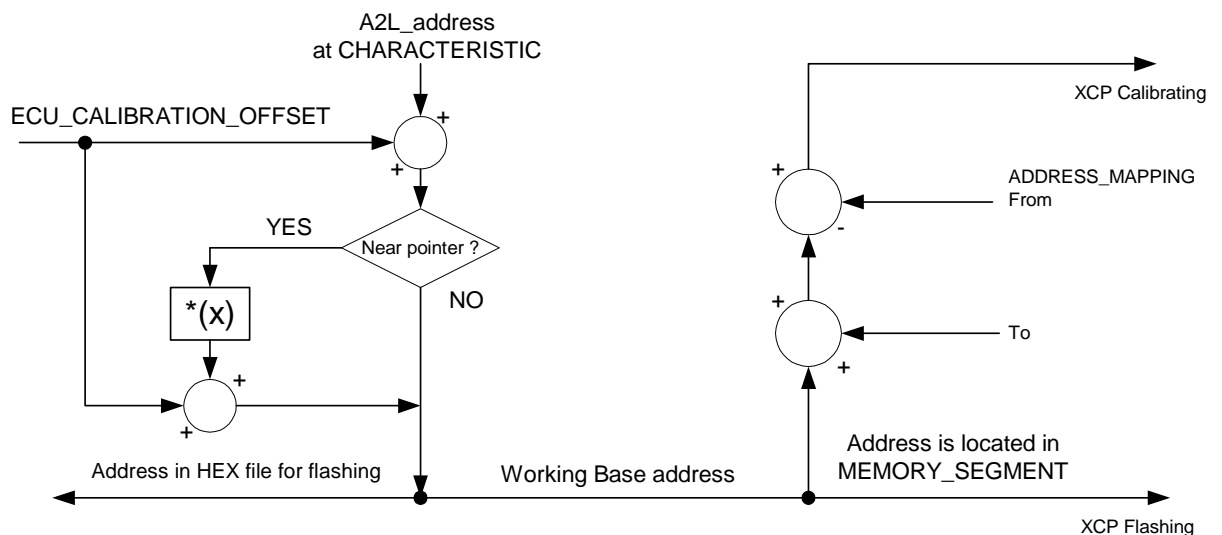


Diagram 14 : address calculation

1.4.3.2 MASTER - SLAVE

The slave has to support checksum calculation on all address ranges that are described with SECTORS or SEGMENTS.

Checksum calculation has to be possible for all PAGEs that have XCP_ACCESS_ALLOWED.

If a PAGE is READABLE by the XCP master, the master can access this PAGE with the commands UPLOAD and SHORT_UPLOAD, in standard mode and if supported in block mode.

If a PAGE is WRITEABLE by the XCP master, the master can access this PAGE with the commands SHORT_DOWNLOAD and DOWNLOAD_MAX in standard mode.

If a PAGE is WRITEABLE by the XCP master, the master can access this PAGE with the commands DOWNLOAD and if block mode supported with DOWNLOAD_NEXT.

If a PAGE is WRITEABLE by the XCP master, the master can access this PAGE with the command MODIFY_BITS which allows to modify bits in an atomic way.

1.4.3.3 PAGE – PAGE

If the XCP slave device has more than one PAGE, the master can copy the data from one PAGE to another with COPY_CAL_PAGE.

In principal any PAGE of any SEGMENT can be copied to any PAGE of any SEGMENT. However, restrictions might be possible. The slave indicates this by ERR_PAGE_NOT_VALID, ERR_SEGMENT_NOT_VALID or ERR_WRITE_PROTECTED.

1.5 FLASH PROGRAMMING

1.5.1 THE FLASHING MODEL

1.5.1.1 PHYSICAL LAY-OUT: SECTOR

The physical lay-out of the slave's memory is described with objects called SECTORS. SECTOR start addresses and sizes are important when reprogramming (Flashing) the slave device.

A SECTOR is referenced by a SECTOR_NUMBER.

Within one and the same XCP slave device, the range for the SECTOR_NUMBER starts from 0 and has to be continuous.

SECTOR_NUMBER [0,1,..255]

1.5.1.2 GENERAL:

In principle the complete flash process can be divided into three steps. It depends on the point of view, whether the individual use case needs all of them:

- administration before (for example version control)
- original flash process ('only' the programming actions)
- administration below (for example version or checksum control)

The XCP protocol deals with these steps in different ways. The commands for the original flash process are the focus of XCP.

XCP offers special programming commands. The project specific use of all the commands must be specified in a project specific "programming flow control". This document specifies no standard for this additional description file. In practice every project needs a project specific agreement between the ECU and the tool supplier.

List without any sequence definition:

```
PROGRAM_START
PROGRAM_CLEAR
PROGRAM_FORMAT
PROGRAM          (Loop)
It is also possible to use a block transfer mode optionally.
PROGRAM_VERIFY
PROGRAM_RESET
```

Usually administration before means version control before the original flash process has been started. This examination checks inside the tool whether the new flash content fits to the ECU. Therefore the tool needs identification information of the ECU and of the new flash content. XCP doesn't support special version control commands for the flash process. In practice the administration actions are very project specific and it depends on the ECU, which services are necessary.

The ECU functional description can specify with which standard XCP commands a version control before could be done.

The actions of the version control below can be done inside the ECU. XCP supports some flexible commands.

The original flash process can be done with different concepts. The XCP protocol supports two different flash access methods. They are called the "absolute" and the "functional" access modes. Both methods use the same commands with sometimes different parameters. It is possible to mix, i.e. to use a different access method for the delete phase in comparison to the programming phase. The recommended concept is based on the available address and memory information and specified in the project specific programming flow control.

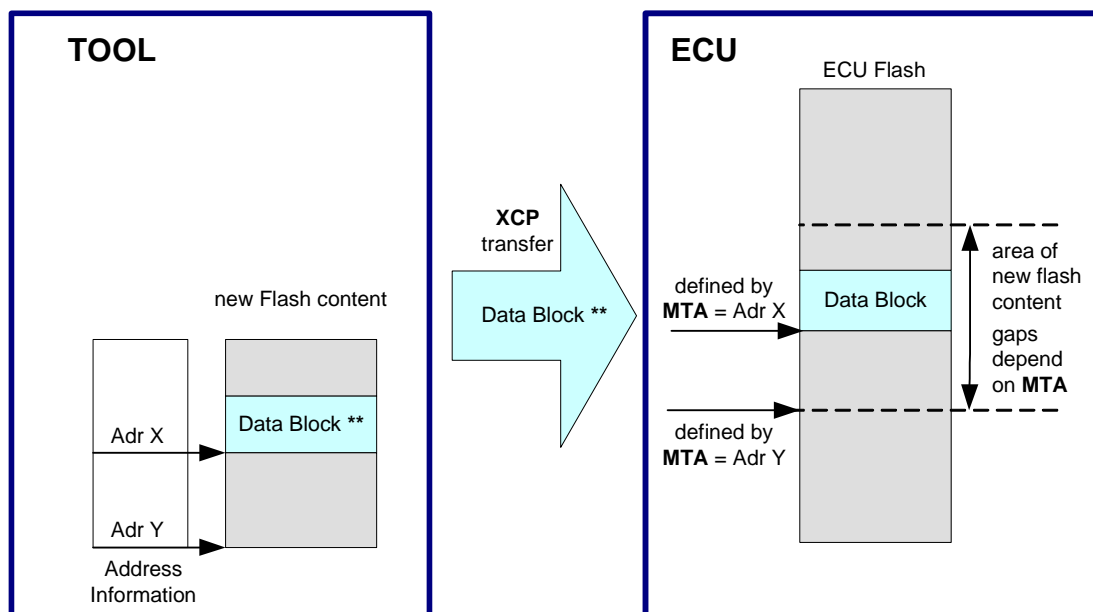
1.5.1.3 ABSOLUTE ACCESS MODE : ACCESS BY ADDRESS

This mode bases on some conditions and is used as default. The physical layout of the flash device is well-known to the tool and the flash content to be programmed is available and also the address information of the data.

It depends on the project, whether the physical layout information are supported by an description file or can be read out of the ECU. There exist different optional XCP commands for different information.

Moreover the tool needs all the necessary sequence information, which must be specified in a project specific programming flow control.

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.



LEGEND

Data Block ** : may be crypted and/or compressed

XCP Transfer : flash gaps (= missing addresses inside area of new flash content) are allowed

Diagram 15 : Absolute Access Mode

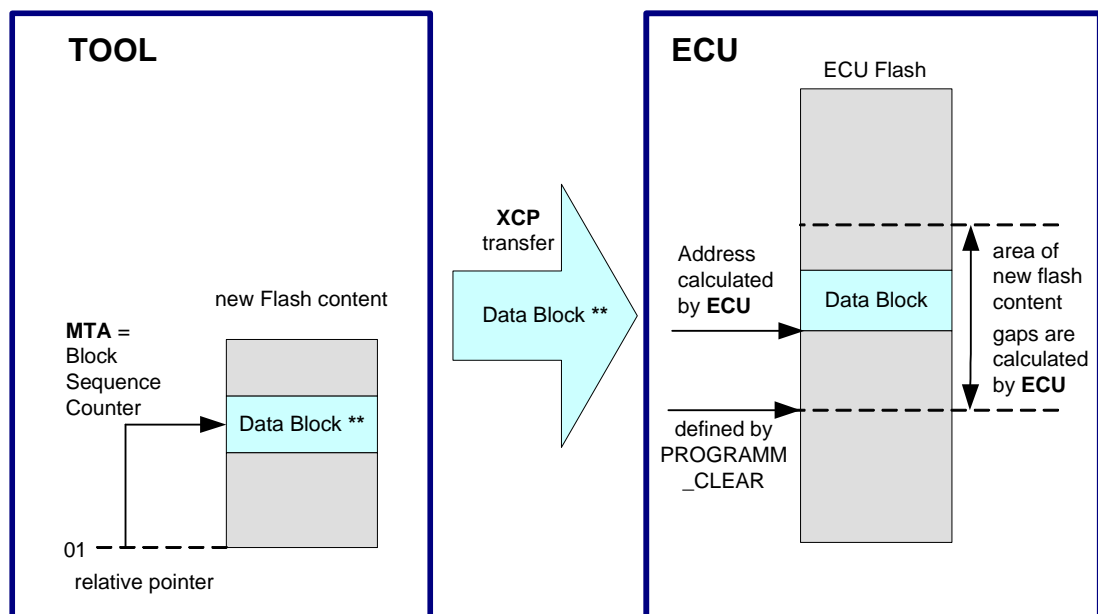
1.5.1.4 FUNCTIONAL ACCESS MODE : ACCESS BY FLASH AREA

This mode is suitable for two different use-cases. The tool needs no memory mapping information and no address information of the flash content to be programmed

The tool needs only the information about the flash area and uses the address information in a different way. The address information represents a relative pointer related to the download software and starts with zero. This mode is useful in connection with compressed or encrypted download software. In this use-case there is no direct relationship between a physical address and the content behind.

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.



LEGEND

Data Block ** : may be crypted and/or compressed

XCP Transfer : missing relative pointer during XCP transfer are not allowed

Diagram 16 : Functional Access Mode

The behaviour is similar to ISO 14229-1 (Road vehicles - Diagnostic services - Part 1: Specification and requirements) and ISO 15765-3 (Road vehicles - Diagnostics on controller area network (CAN) - Part 3: Implementation of diagnostic services) .

If a PROGRAM request is correctly received and processed in the slave, but the positive response message does not reach the master, then the master would determine an application layer timeout and would repeat the same request (including the same Block Sequence Counter). The slave would receive the repeated PROGRAM request and could determine based on the included Block Sequence Counter, that this PROGRAM request is repeated. The slave would send the positive response message immediately without writing the data once again into its memory.

If the PROGRAM request is not received correctly in the slave, then the slave would not send a positive response message. The master would determine an application layer timeout and would repeat the same request (including the same Block Sequence Counter). The slave would receive the repeated PROGRAM request and could determine based on the included Block Sequence Counter that this is a new PROGRAM request. The slave would process the service and would send the positive response message.

It is optionally possible to change to the absolute access mode at the end of the flash session.

Affected Commands

PROGRAM_CLEAR, PROGRAM_FORMAT, PROGRAM, SET_MTA

1.5.1.5 CHECKSUM CONTROL AND PROGRAM VERIFY

After the original flash process a version control is helpful. This action checks whether the new flash content fits to the rest of the flash. In practice exists different methods, but XCP supports only a checksum control and the start of internal test routines.

The checksum method can be done with the standard checksum command (examination inside the tool). On the other hand XCP supports an examination inside the slave. The tool can start slave internal test routines and send verification values to the slave.

Affected Commands

BUILD_CHECKSUM, PROGRAM_VERIFY

1.5.1.6 END OF FLASH SESSION

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.

Affected Commands
PROGRAM_RESET

2 THE XCP PROTOCOL

2.1 TOPOLOGY

The XCP protocol basically is a single-master/single-slave type of communication. Any communication always is initiated by the master. The slave has to respond upon requests from the master with an appropriate response.

The XCP Protocol uses a “soft” master/slave principle. Once the master established a communication channel with the slave, the slave is allowed to send certain messages (Events, Service Requests and Data Acquisition messages) autonomously. Also the master sends Data stimulation messages without expecting a direct response from the slave.

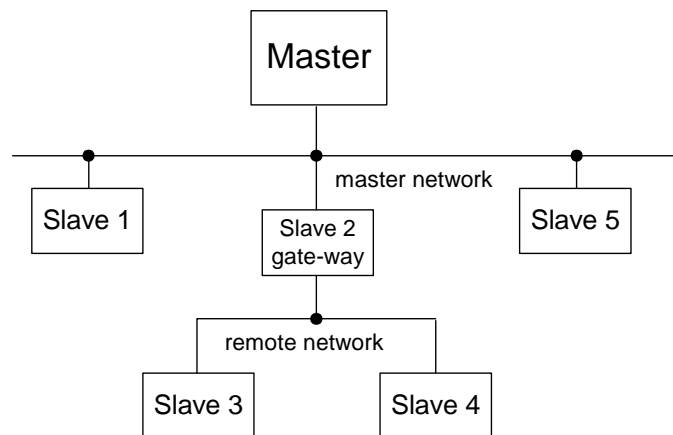


Diagram 17 : The XCP Topology

The master when establishing a communication channel, builds a continuous, logical, unambiguous point-to-point connection with 1 specific slave. A slave device driver cannot handle multiple connections.

The XCP Protocol does not allow a “single-master/multi-slave” topology. The master is not allowed to broadcast XCP messages to multiple slaves at the same time. The only exception is GET_SLAVE_ID on CAN that can be broadcasted.

The XCP Protocol however, allows a “multiple single-master/single-slave” topology. Several “single-master/single-slave” communication channels can be active in the same network at the same time. The identification parameters of the Transport Layer (e.g. CAN identifiers on CAN) have to be chosen in such a way that they build independent and unambiguously distinguishable communication channels.

The XCP Protocol allows gate-ways to be part of the topology. The network the master directly is connected to is called the Master Network. The network the master indirectly, through a gate-way is connected to, is called the Remote Network.

When transferring XCP messages, a gate-way has to be able to adapt the XCP Header and Tail depending upon the Transport Layer used in Master Network and Remote Network.

The XCP gate-way has to logically represent the nodes of its Remote Network in the Master Network.

Example:

Master Network = CAN 500000 bps

Remote Network = CAN 250000 bps

Master with Slave 1

Master sends with CAN-Id = 0x100 on Master Network

Slave 1 sends with CAN-Id = 0x110 on Master Network

Master with Slave 2 (Slave 2 directly)

Master sends with CAN-Id = 0x200 on Master Network

Slave 2 sends with CAN-Id = 0x210 on Master Network

Master with Slave 3 (Slave 2 as gate-way)

Master sends with CAN-Id = 0x300 to Slave 2 on Master Network

Slave 2 sends with CAN-Id = 0x100 to Slave 3 on Remote Network

Slave 3 sends with CAN-Id = 0x110 to Slave 2 on Remote Network

Slave 2 sends with CAN-Id = 0x310 on Master Network

2.2 THE XCP COMMUNICATION MODELS

2.2.1 THE STANDARD COMMUNICATION MODEL

In the connected state, each request packet will be responded by a corresponding response packet or an error packet.

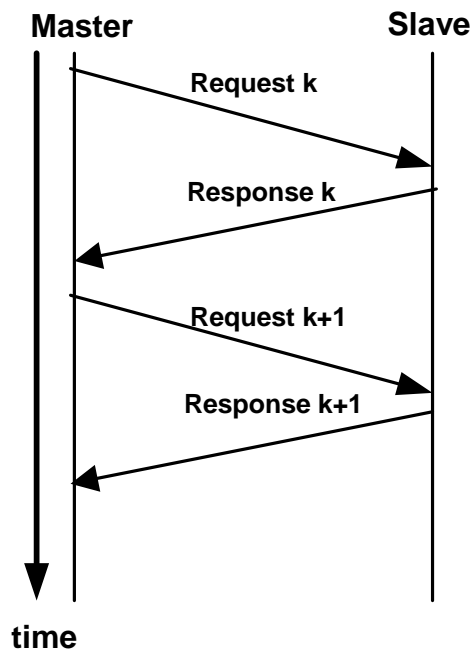


Diagram 18 : Standard Communication Model

In the standard communication model, the master device may not send a new request until the response to the previous request has been received.

2.2.2 THE BLOCK TRANSFER COMMUNICATION MODEL

In XCP Standard Communication mode, each request packet will be responded by a single response packet or an error packet.

To speed up memory uploads, downloads and flash programming, the XCP commands UPLOAD, SHORT_UPLOAD, DOWNLOAD, SHORT_DOWNLOAD and PROGRAM may support a block transfer mode similar to ISO/DIS 15765-2.

The block transfer communication mode excludes interleaved communication mode.

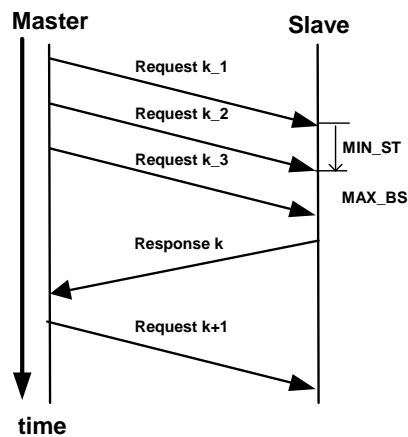


Diagram 19 : Master Block Transfer

MASTER_BLOCK_MODE_SUPPORTED in COMM_MODE_OPTIONAL at GET_COMM_MODE_INFO indicates whether the master may use Master Block Transfer Mode.

The slave device may have limitations for the maximum block size and the minimum separation time. The communication parameters MIN_ST and MAX_BS are obtained by the GET_COMM_MODE_INFO command. It's in the responsibility of the master device to care for the limitations. For details, refer to the description of the DOWNLOAD command.

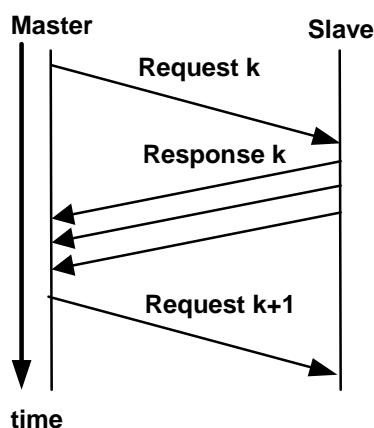


Diagram 20 : Slave Block Transfer

SLAVE_BLOCK_MODE_SUPPORTED in COMM_MODE_BASIC at CONNECT indicates whether the slave supports Slave Block Transfer Mode.

There are no limitations allowed for the master device. The separation time for the subsequent responses may be 0. The master device has to support the maximum possible block size. For details, refer to the description of the UPLOAD command.

2.2.3 THE INTERLEAVED COMMUNICATION MODEL

In the standard communication model, the master device may not send a new request until the response to the previous request has been received.

To speed up data transfer, in Interleaved Communication Mode the master may already send the next request before having received the response on the previous request.

The interleaved communication mode excludes block transfer communication mode.

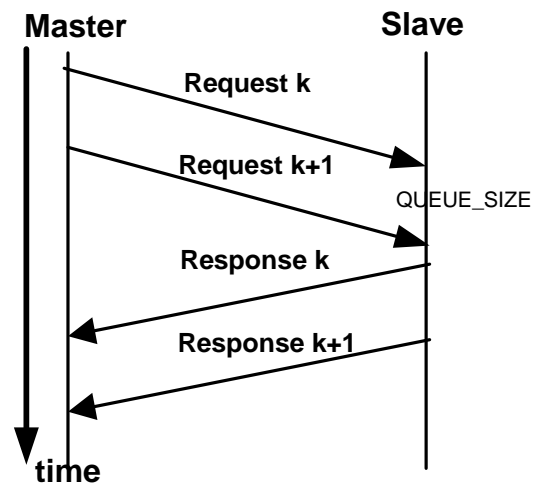


Diagram 21 : Interleaved Communication Model

INTERLEAVED_MODE_SUPPORTED at GET_COMM_MODE_INFO indicates whether the master may use Interleaved Mode.

The slave device may have limitations for the maximum number of consecutive requests it can buffer. The communication parameter QUEUE_SIZE is obtained by the GET_COMM_MODE_INFO command. It's in the responsibility of the master device to care for this limitation.

2.3 STATE MACHINE

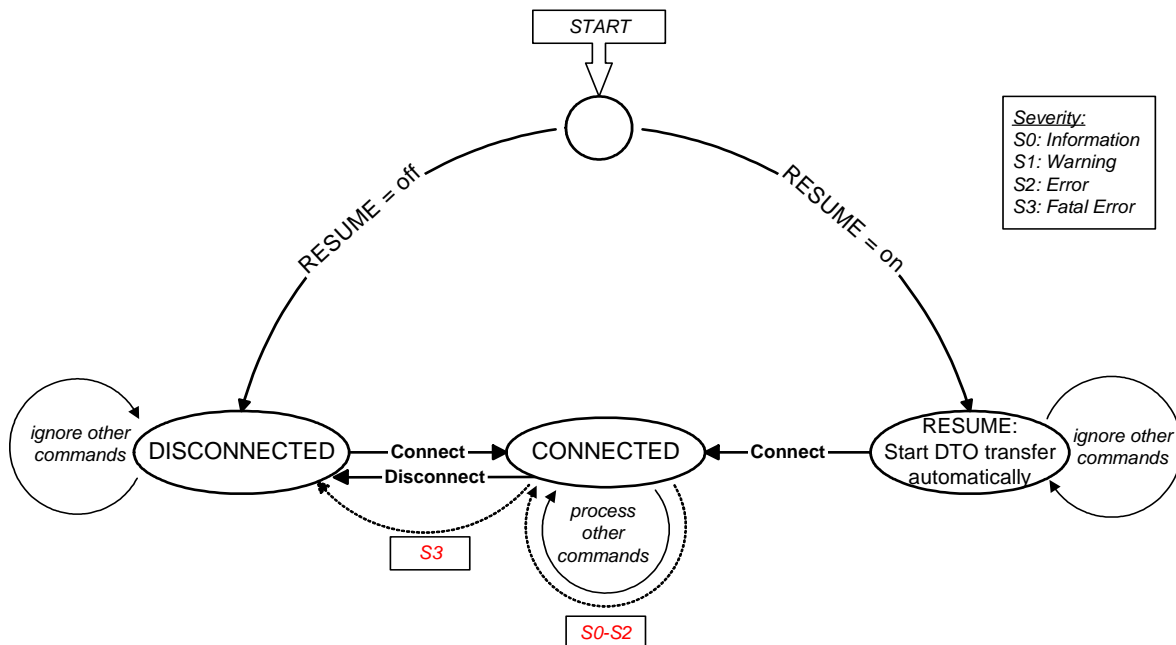


Diagram 22 : The XCP slave State Machine

As soon as the XCP slave device starts its operation, it has to check whether there's a DAQ list configuration, to be used for RESUME mode, available in non-volatile memory.

If there's no such a configuration available, the slave has to go to "DISCONNECTED" state.

In "DISCONNECTED" state, there's no XCP communication. The session status, all DAQ lists and the protection status bits are reset, which means that DAQ list transfer is inactive and the seed and key procedure is necessary for all protected functions.

In "DISCONNECTED" state, the slave processes no XCP commands except for CONNECT.

On CAN the slave additionally to CONNECT also will accept a GET_SLAVE_ID.

The CONNECT command establishes a **continuous, logical, point-to-point connection** with the slave and brings the slave in a "CONNECTED" state.

In "CONNECTED" state, the slave processes any XCP command packet by responding with a corresponding response packet or an error packet.

With a `CONNECT(Mode = USER_DEFINED)`, the master can start an XCP communication with the slave and at the same time tell the slave that it should go into a special (user defined) mode, which has no influence on the behavior of the XCP driver of the slave.

For a `CONNECT(USER_DEFINED)` command, the normal Time-Out Handling rules do not apply. The master continuously has to send a `CONNECT(USER_DEFINED)` to the slave until he receives an acknowledgment. The master has to use the time-out value `t6` between the commands. The master just has to repeat the `CONNECT(USER_DEFINED)` without any `SYNCH`, `Pre-action` or `Action`.

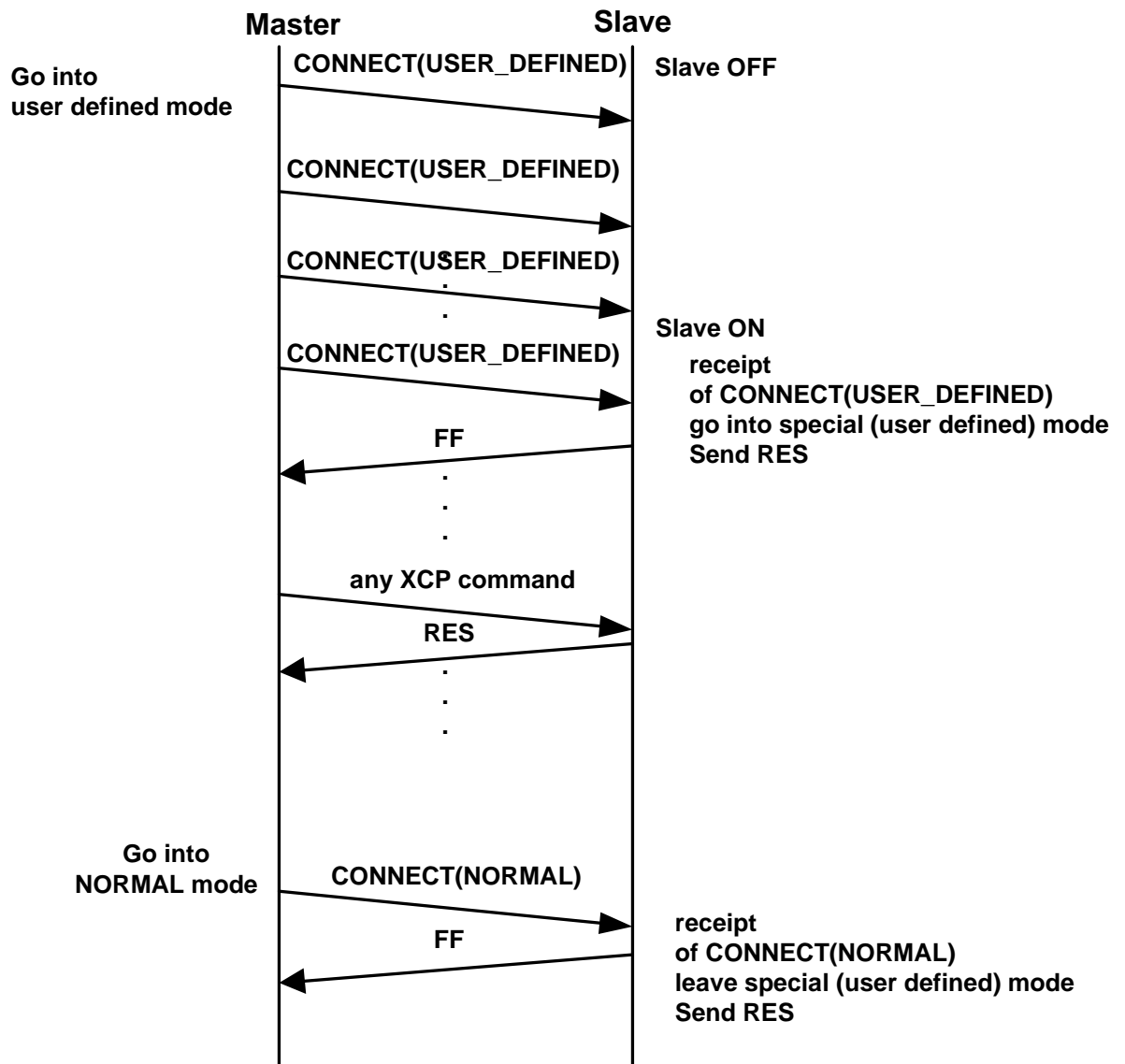


Diagram 23 : Typical use of `CONNECT` modes `USER_DEFINED` and `NORMAL`

With a CONNECT(Mode = NORMAL), the master can start an XCP communication with the slave.

In "CONNECTED" state, the slave has to acknowledge a new CONNECT and handle it like a CONNECT command to a disconnected device.

If the slave when starting its operation detects that there's a DAQ list configuration, to be used for RESUME mode, available in non-volatile memory, the slave has to go to the "RESUME" state.

In "RESUME", the slave automatically has to start those DAQ lists that are stored in non-volatile memory and that are to be used for RESUME mode (ref. Description of RESUME mode).

In "RESUME", the slave processes no XCP commands except for CONNECT.

In "RESUME" state, the slave has to acknowledge a CONNECT and handle it like a CONNECT command to a disconnected device, but keep the current DTO transfer running.

In "CONNECTED" state, if the master sends a DISCONNECT command, the slave goes to "DISCONNECTED" state.

If an error occurs with severity S0-S2, the slave will not change its state.

If an error occurs with severity S3 "Fatal error", this will bring the slave to the "DISCONNECTED" state.

2.4 PROTECTION HANDLING

XCP provides protection handling for the features

- measurement / stimulation
- calibration
- flashing

The concept is to use in advance a command to exchange a seed and a key value. The key length is big enough to support also asymmetrical algorithms. If the corresponding security access algorithm was successfully computed by the XCP master, the XCP slave allows access to the requested XCP commands. For more details please look at the following commands:

- GET_STATUS
- GET_SEED
- UNLOCK

Moreover it could be necessary to protect the software itself regarding reading. The need for information hiding can be different depending on the project phase (development or after-sales) and is implementation specific.

The following commands are suitable to read memory contents:

- UPLOAD
- SHORT_UPLOAD
- BUILD_CHECKSUM

Due to the fact that these commands cannot be protected with the standard security mechanism, a different method is specified. If the XCP slave decides internally to hide information, it will answer with the negative response `ERR_ACCESS_DENIED`. This response indicates (in contrast to `ERR_ACCESS_LOCKED`) that the XCP master cannot unlock the requested command.

Remark:

In any case it must be possible to read out ID information of the XCP slave (requested with `GET_ID`) if an XCP master needs this information for continuation.

2.5 THE XCP MESSAGE (FRAME) FORMAT

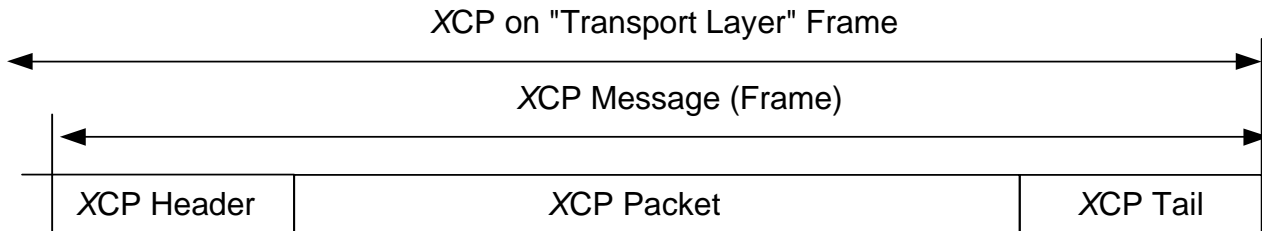


Diagram 24 : The XCP Message (Frame) format

XCP messages always are transported in the Data Field of a particular transport layer e.g. CAN, TCP/IP and UDP/IP. In general, the transport layer has to fulfill the requirements below :

- the length and the contents of a message may not change
- the sequence of the messages may not change
- messages may not be duplicated

An XCP Message (= Frame) consists of an XCP Header, an XCP Packet and an XCP Tail.

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

An XCP Packet consists of an Identification Field, an optional Timestamp Field and a Data Field. Part 2 of the XCP Specification, "Protocol Layer Specification", describes the contents of an XCP Packet.

The XCP Header and XCP Tail depend upon the transport layer used.

Both XCP Header and XCP Tail consist of a Control Field.

Part 3 of the XCP Specification, "Transport Layer Specification", describes the contents of the Control Fields for different transport layers e.g. CAN, TCP/IP and UDP/IP.

3 THE LIMITS OF PERFORMANCE

3.1 GENERIC PERFORMANCE PARAMETERS

MAX_CTO shows the maximum length of a CTO packet in bytes.
MAX.DTO shows the maximum length of a DTO packet in bytes.

Name	Type	Representation	Range of value
MAX_CTO	Parameter	BYTE	0x08 – 0xFF
MAX.DTO	Parameter	WORD	0x0008 – 0xFFFF

The range of both protocol parameters can be smaller, depending on the used transport layer.

3.2 DAQ/STIM SPECIFIC PERFORMANCE PARAMETERS

MAX_EVENT_CHANNEL indicates the number of event channels on the XCP slave.

An event channel is identified by a number called EVENT_CHANNEL_NUMBER.

Name	Type	Representation	Range of value
MAX_EVENT_CHANNEL	Parameter	WORD	0x0000 – 0xFFFF
MAX_EVENT_CHANNEL_ABS	Constant	WORD	0xFFFF
EVENT_CHANNEL_NUMBER	Parameter	WORD	0x0000 – 0xFFFE
EVENT_CHANNEL_NUMBER_MAX	Parameter	WORD	MAX_EVENT_CHANNEL – 1
EVENT_CHANNEL_NUMBER_MAX_ABS	Constant	WORD	0xFFFE

MAX_DAQ indicates the number of DAQ lists on the XCP slave.

A DAQ list is identified by a number called DAQ_LIST_NUMBER.

Counting starts at zero.

MIN_DAQ indicates the number of predefined, read only DAQ lists on the XCP slave.

DAQ_COUNT indicates the number of DAQ lists for dynamic configuration.

Name	Type	Representation	Range of value
MAX_DAQ	Parameter	WORD	0x0000 – 0xFFFF
MAX_DAQ_ABS	Constant	WORD	0xFFFF
DAQ_COUNT	Parameter	WORD	0x0000 – 0xFFFF
MIN_DAQ	Parameter	BYTE	0x00 – 0xFF
DAQ_LIST_NUMBER	Parameter	WORD	0x0000 – 0xFFFE

MAX_ODT_ENTRIES indicates the maximum amount of entries into an ODT of the XCP slave.

ODT_ENTRIES_COUNT indicates the amount of entries into an ODT using dynamic DAQ list configuration.

An entry is identified by a number called ODT_ENTRY_NUMBER.

Counting starts at zero.

Name	Type	Representation	Range of value
MAX_ODT_ENTRIES	Parameter	BYTE	0x00 – 0xFF
ODT_ENTRIES_COUNT	Parameter	BYTE	0x00 – 0xFF
ODT_ENTRY_NUMBER	Parameter	BYTE	0x00 – 0xFE

3.2.1 DAQ SPECIFIC PARAMETERS

MAX_ODT indicates the maximum amount of ODTs of the XCP slave.

MAX_ODT_ENTRY_SIZE_DAQ indicates the upper limit for the size of the element described by an ODT entry.

ODT_COUNT indicates the amount of ODTs of a DAQ list using dynamic DAQ list configuration.

An ODT is identified by a number called ODT_number.

Counting starts at zero.

Name	Type	Representation	Range of value
MAX_ODT	Parameter	BYTE	0x00 – 0xFC
MAX_ODT_ENTRY_SIZE_DAQ	Parameter	BYTE	0x00 – 0xFF
ODT_COUNT	Parameter	BYTE	0x00 – 0xFC
ODT_NUMBER	Parameter	BYTE	0x00 – 0xFB

3.2.2 STIM SPECIFIC PARAMETERS

MAX_ODT indicates the maximum amount of ODTs of the XCP slave.

MAX_ODT_ENTRY_SIZE_STIM indicates the upper limit for the size of the element described by an ODT entry.

ODT_COUNT indicates the amount of ODTs of a DAQ list using dynamic DAQ list configuration.

An ODT is identified by a number called ODT_number.

Counting starts at zero.

Name	Type	Representation	Range of value
MAX_ODT	Parameter	BYTE	0x00 – 0xC0
MAX_ODT_ENTRY_SIZE_STIM	Parameter	BYTE	0x00 – 0xFF
ODT_COUNT	Parameter	BYTE	0x00 – 0xC0
ODT_NUMBER	Parameter	BYTE	0x00 – 0xBF

4 VERSIONING

4.1 THE XCP PROTOCOL LAYER VERSION NUMBER

Part 2 of the XCP Specification, "Protocol Layer Specification", describes the contents of an XCP Packet. The XCP Packet is the generic part of the protocol that is independent from the Transport Layer used.

The XCP Protocol Layer Version Number indicates the version of the Protocol Layer Specification. The XCP Protocol Layer Version Number is a 16-bit value.

If the Protocol Layer is modified in such a way that a functional modification in the slave's driver software is needed, the higher byte of the XCP Protocol Layer Version Number will be incremented.

This could be the case e.g. when modifying the parameters of an existing command or adding a new command to the specification.

If the Protocol Layer is modified in such a way that it has no direct influence on the slave's driver software, the lower byte of the XCP Protocol Layer Version Number will be incremented.

This could be the case e.g. when rephrasing the explaining text or modifying the AML description.

The slave only returns the most significant byte of the XCP Protocol Layer Version Number in the response upon CONNECT.

4.2 THE XCP TRANSPORT LAYER VERSION NUMBER

Part 3 of the XCP Specification, "Transport Layer Specification", describes the contents of the Control Fields for different transport layers e.g. CAN, TCP/IP and UDP.

Part 3 of the XCP Specification also describes possible additional Packets for a specific Transport Layer.

Independent from Part 2, every Part 3 has its own XCP Transport Layer Version Number.

The XCP Transport Layer Version Number indicates the version of a specific Part 3 of the specification.

The XCP Transport Layer Version Number is a 16-bit value.

If a specific Part 3 is modified in such a way that a functional modification in the slave's driver software is needed, the higher byte of its XCP Transport Layer Version Number will be incremented. This could be the case e.g. when modifying the parameters of an existing command or adding a new command to the specification.

If a specific Part 3 is modified in such a way that it has no direct influence on the slave's driver software, the lower byte of its XCP Transport Layer Version Number will be incremented. This could be the case e.g. when rephrasing the explaining text or modifying the AML description.

The slave only returns the most significant byte of the XCP Transport Layer Version Number for the current Transport Layer in the response upon CONNECT.

4.3 THE COMPATIBILITY MATRIX

The main.a2l that describes a slave that supports XCP on different Transport Layers, includes an **XCP_definitions.a2l** that contains a reference to a certain version of Protocol Layer Specification and (a) reference(s) to (a) certain version(s) of Transport Layer Specification(s).

If a certain version of the Protocol Layer Specification needs a certain version of a Transport Layer Specification, this will be mentioned as prerequisite in the Protocol Layer Specification.

If a certain version of a Transport Layer Specification needs a certain version of Protocol Layer Specification, this will be mentioned as prerequisite in the Transport Layer Specification.

The Compatibility Matrix at

www.asam.net \ ... \ Current specifications

gives an overview of the allowed combinations of XCP Protocol Layer Version Number and XCP Transport Layer Version Number.

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

Tel.: (+49) 08102 / 8953 17
Fax.: (+49) 08102 / 8953 10
E-mail: info@asam.net
Internet: www.asam.net