ASAM AE MCD-1 XCP

Universal Measurement and Calibration Protocol

Protocol Layer Specification

Version 1.2

Date: 2013-06-20

Base Standard

Association for **S**tandardization of
Automation and **M**easuring Systems

Status of Documents

Date:	2013-06-20
Author:	ASAM e.V.
Version:	1.2
Status:	Release

Disclaimer

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.
Without explicit authorization by ASAM e. V. this document may neither be changed nor in the original or a changed state be implemented / integrated in other documents. This also applies for parts of this document.

Table of Contents

1 FOREWORD

XCP is short for Universal Measurement and Calibration Protocol. The main purpose is the data acquisition and calibration access from electronic control units. Therefore a generic protocol layer is defined. As transport medium different physical busses and networks can be used. For each authorized transport medium a separate transport layer is defined. This separation is reflected in standard document structure, which looks like follows:

- One Base Standard
- Associated Standards for each physical bus or network type

The Base standard describes the following content:

- Protocol Layer
- Interface to ASAM MCD-2 MC
- Interface to an external SEED&KEY function
- Interface to an external Checksum function
- Interface to an external A2L Decompression/Decrypting function
- Example Communication sequences

For each transport layer exist an own associated standard. For the version in hand the following transport layers are defined

- XCP on CAN
- XCP on Ethernet (TCP/IP, UDP/IP)
- XCP on SxI (SPI, SCI)
- XCP on USB
- XCP on Flexray

The "X" inside the term XCP generalizes the "various" transportation layers that are used by the members of the protocol family. Because XCP is based on CCP the "X" shall also show that the XCP protocol functionality is extended compared to CCP.

2 INTRODUCTION

XCP shall be used in all stages of ECU development, like

- ECU development
- ECU testing
- Bypass usage for Rapid Control Prototyping systems (function development)

Beside measurement data acquisition and calibration process XCP is also used for flashing and inside the hardware in the loop simulation. For each ECU exists a description file, which includes all necessary elementary data about measurements and characteristics, and includes all descriptive data like addresses, data types, dimensions [1].

XCP was designed according to the following principles:

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

XCP is designed as single master multi slave concept, and supports the following basic and optional features:

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

Optional features:

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

XCP uses no ASAM data types, because the transport of memory segments takes place via the different transport layers. ASAM data types are used in the respective interfaces, which uses the data like described in the a2l description files. On this level the native data convert into ASAM data types.

This base standard starts with a description of an XCP Features Overview. Also The Limits of Performance are shown. The XCP protocol consists of an XCP packet, an XCP

Header and an XCP Tail. Header and Tail are dependent from used transport layer and therefore not described inside this base standard.

3 SYMBOL AND ABBREVIATED TERMS

The following terms and abbreviations are used within the document:

Abbreviation	Description
A2L	File Extension for an **A**SAM **2**MC **L**anguage File
AG	**A**ddress **G**ranularity
AML	**A**SAM 2 **M**eta **L**anguage
CAL	**CAL**ibration
CAN	**C**ontroller **A**rea **N**etwork
CCP	**C**AN **C**alibration **P**rotocol
CMD	**C**o**M**man**D**
CTO	**C**ommand **T**ransfer **O**bject
DAQ	**D**ata **A**c**Q**uisition, **D**ata **A**c**Q**uisition Packet
DLL	**D**ynamically **L**inked **L**ibrary
DTO	**D**ata **T**ransfer **O**bject
ECU	**E**lectronic **C**ontrol **U**nit
ERR	**ERR**or Packet
EV	**EV**ent Packet
ID	**Id**entifier
IF	**I**nter**F**ace
IP	**I**nternet **P**rotocol
LSB	**L**east **S**ignificant **B**it
MCD	**M**easurement **C**alibration and **D**iagnostics
MSB	**M**ost **S**ignificant **B**it
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
RAM	**R**andom **A**ccess **M**emory
RES	command **RES**ponse packet
ROM	**R**ead-**O**nly **M**emory
SCI	**S**erial **C**ommunication **I**nterface
SERV	**SERV**ice request packet
SO	**S**hared **O**bject
SPI	**S**erial **P**eripheral **I**nterface

Abbrevation	Description
STIM	Data **STIM**ulation packet
TCP	**T**ransfer **C**ontrol **P**rotocol
TCP/IP	**T**ransfer **C**ontrol **P**rotocol **/ I**nternet **P**rotocol
TS	**T**ime **S**tamp
UDP	**U**ser **D**atagram **P**rotocol
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

4 Compatibility

4.1 The XCP Protocol Layer Version Number

This base standard 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 is a 16-bit value, where the high byte contains the major version (X) and low byte contains the minor version (Y) number.

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 mandatory 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 CCP and XCP

XCP is not backwards compatible to an existing CCP implementation.

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

4.3 The Compatibility Matrix

The main.a2l that describes a slave that supports XCP on different Transport Layers, including an **XCP_definitions.aml** 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). For details of the references see chapter Interface to ASAM MCD-2 MC Description File.

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

Table 1 Compatibility matrix

XCP		_on_##	CAN				SxI				TCP/IP and UDP/IP				FlexRay				USB			
_common		U	01				01				01				01				01			
X	Y	V	00	01	02	…	00	01	02	…	00	01	02	…	00	01	02	…	00	01	02	…
01	00		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	
	01		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	
	02		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	
	…																					

5 XCP FEATURES OVERVIEW

5.1 SYNCHRONOUS DATA TRANSFER

5.1.1 DAQ, STIM AND ODT

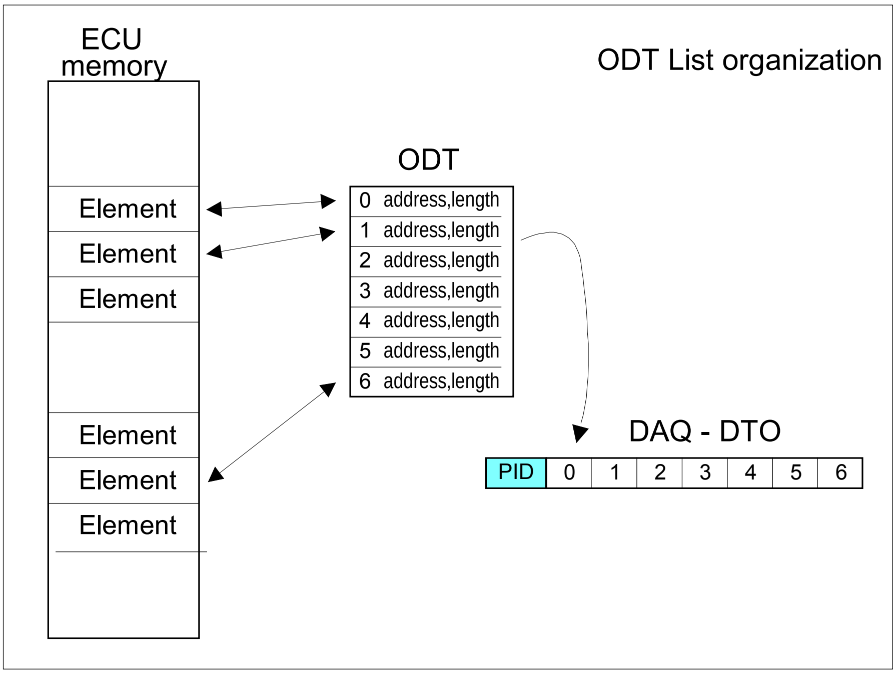

Figure 1 ODT list organization

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 Descriptor 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.

5.1.2 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.
GRANULARITY_ODT_ENTRY_SIZE_x[BYTE] >= 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:

```
Address[AG] mod (GRANULARITY_ODT_ENTRY_SIZE_x[BYTE] / AG[BYTE]) = 0
```

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

```
SizeOf(element described by ODT entry)[AG] mod (GRANULARITY_ODT_ENTRY_SIZE_x[BYTE] / 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:

```
GRANULARITY_ODT_ENTRY_SIZE_x[BYTE] mod (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:

```
SizeOf(element described by ODT entry)[AG] <= 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.

5.1.3 OBJECT DESCRIPTOR TABLE

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]
```

5.1.4 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).

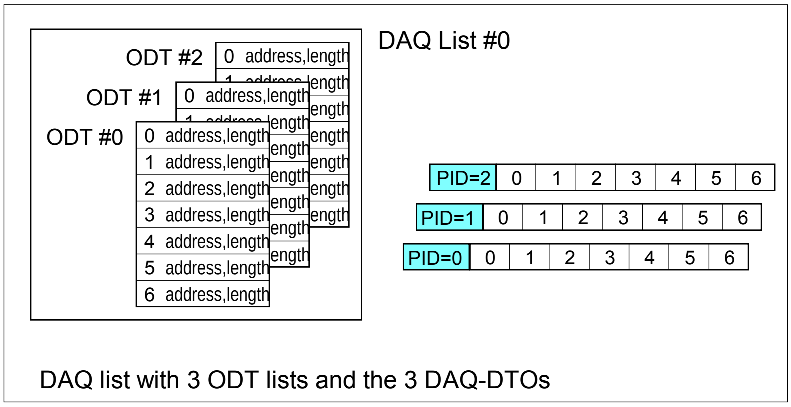

Figure 2 DAQ list organization

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 and has to be continuous.

```
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.

5.1.5 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.

For event channels which have no constant cycle time, e.g. sporadic or crank synchronous events, it is possible to add a minimum cycle time (MIN_CYCLE_TIME), so that the XCP master can make a worst case calculation for e.g. CPU load or required transport layer bandwidth.

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 cannot be used. MAX_DAQ_LIST = 0xFF means there is 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 locally 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 locally at the MEASUREMENT. In case these such lists do 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.

5.1.6 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.

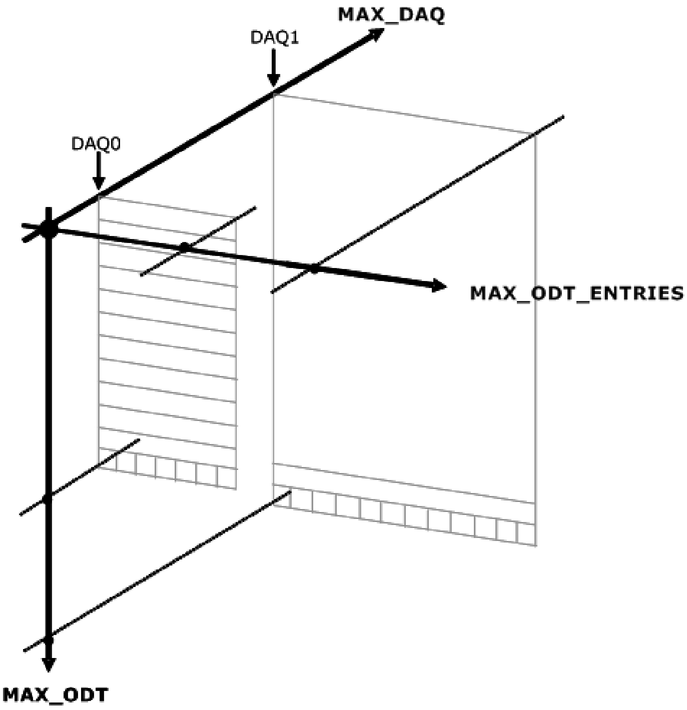

Figure 3 Static DAQ list configuration

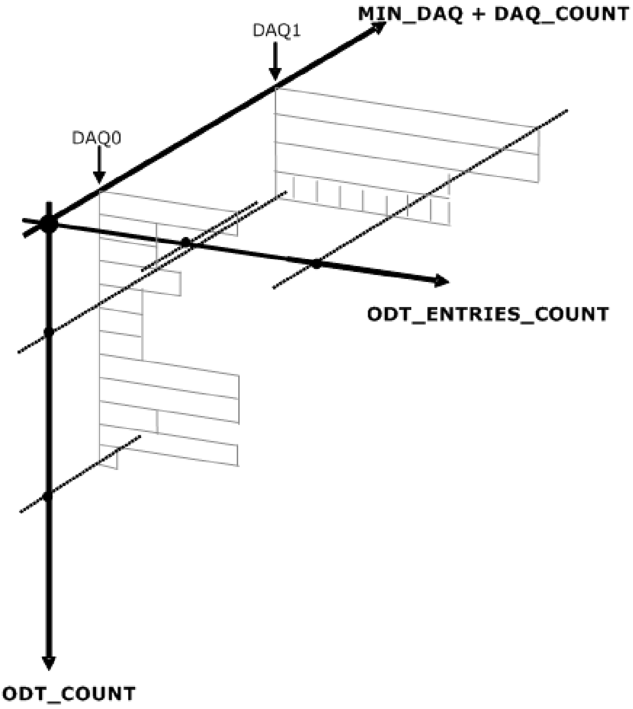

Figure 4 Dynamic DAQ list configuration

If DAQ lists are configured dynamically, other limits apply:

Table 2 DAQ configuration limits of XCP IF_DATA

Static	Dynamic
MAX_DAQ	MIN_DAQ+DAQ_COUNT
	MAX_DAQ_TOTAL
MAX_ODT	MAX_ODT_DAQ_TOTAL
	MAX_ODT_STIM_TOTAL
MAX_ODT_ENTRIES	MAX_ODT_ENTRIES_TOTAL
	MAX_ODT_ENTRIES_DAQ_TOTAL
	MAX_ODT_ENTRIES_STIM_TOTAL

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.

If a parameter is not defined in the IF_DATA XCP, this means that the maximum limit of the appropriate protocol constant applies, see Table 7.

MAX_DAQ_TOTAL specifies the maximum number of dynamic DAQ lists for all DAQ events.
MAX_ODT_DAQ_TOTAL specifies the maximum number of all ODTs having direction DAQ.
MAX_ODT_STIM_TOTAL specifies the maximum number of all ODTs having direction STIM.
MAX_ODT_ENTRIES_TOTAL must be the sum of MAX_ODT_ENTRIES_DAQ_TOTAL and MAX_ODT_ENTRIES_STIM_TOTAL, if these parameters are specified.

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 is 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.

Table 3 DAQ allocation command sequence

		Second Command			
		FREE_DAQ	**ALLOC_DAQ**	**ALLOC_ODT**	**ALLOC_ODT_ ENTRY**
First Command	**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.

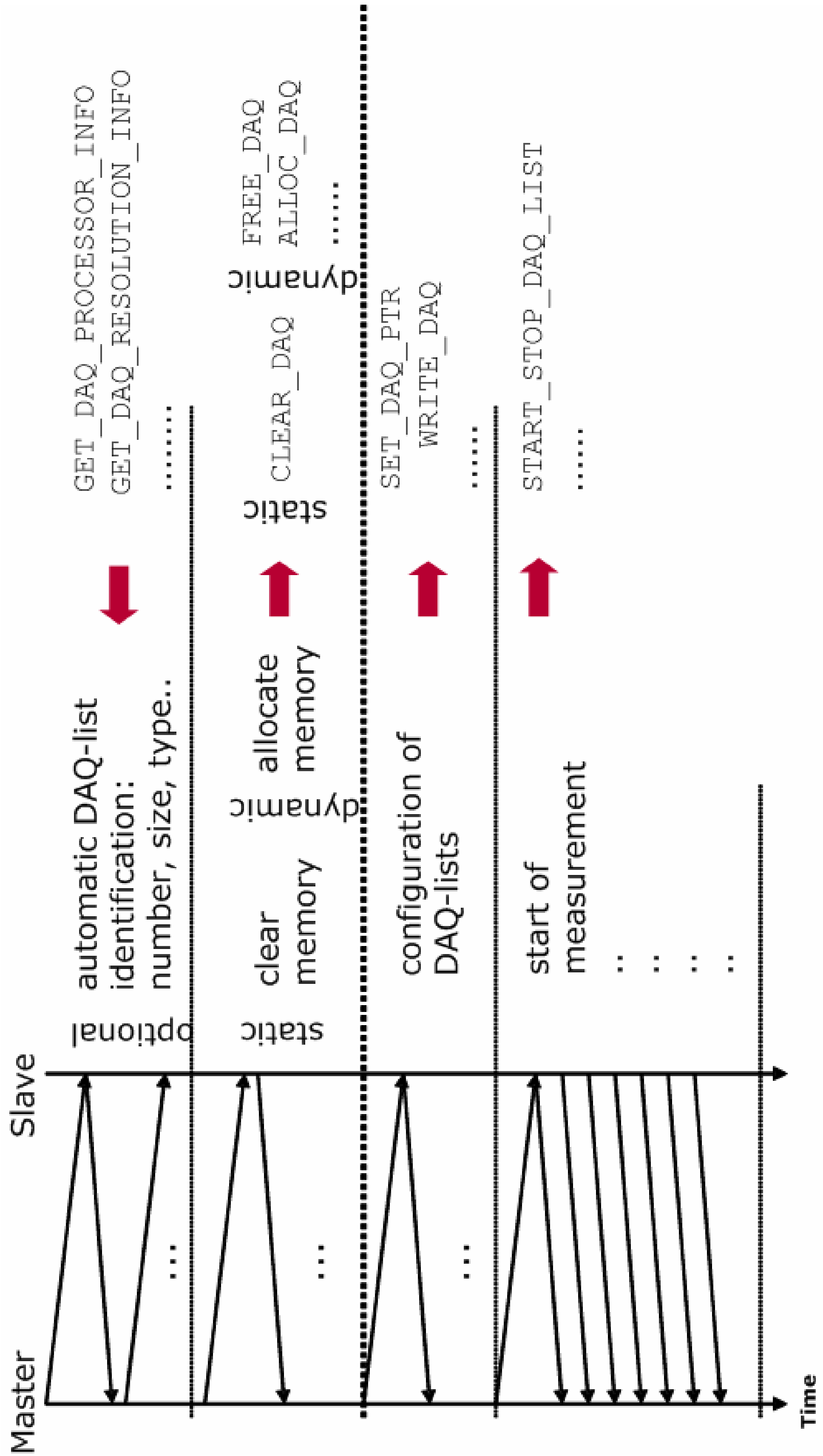

Figure 5 **Static and dynamic DAQ configuration sequence**

5.1.7 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, 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.

5.1.7.1 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.

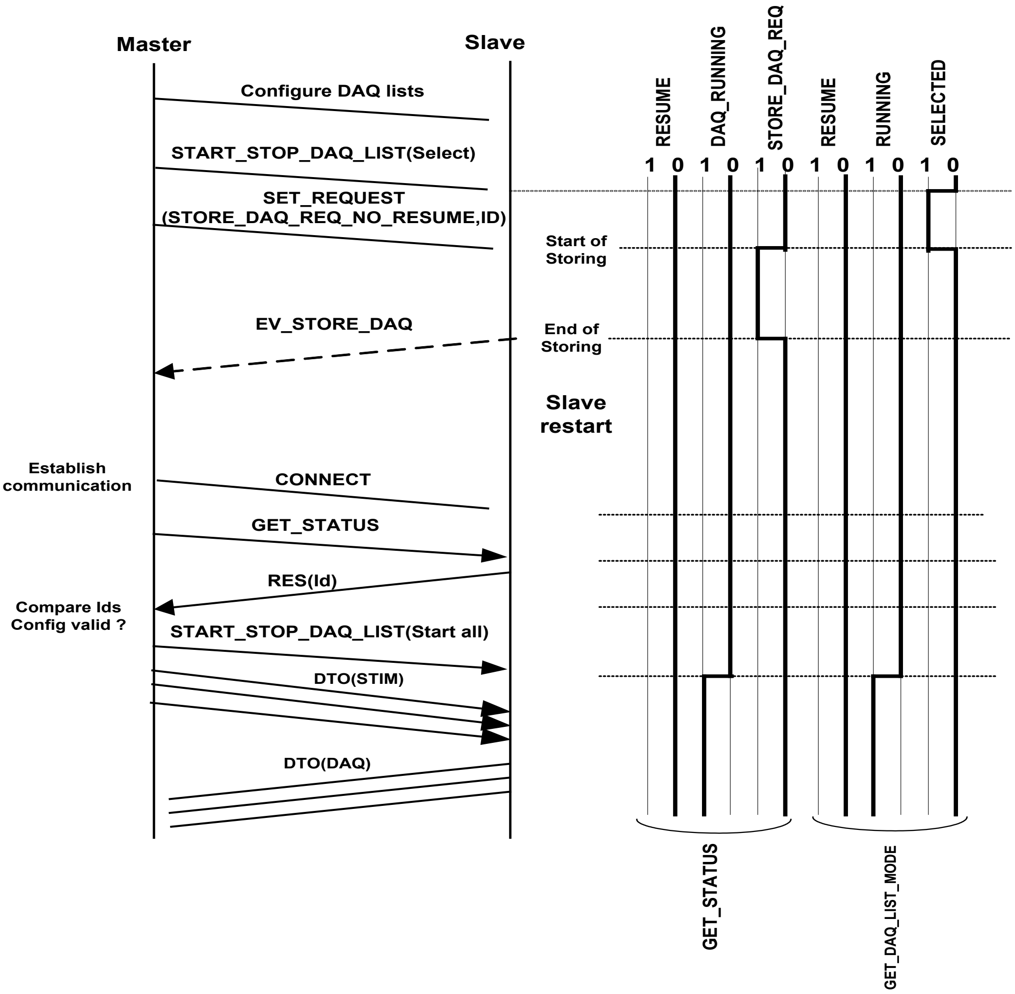

Figure 6 **DAQ configuration storing without power-up data transfer**

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.

5.1.7.2 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.

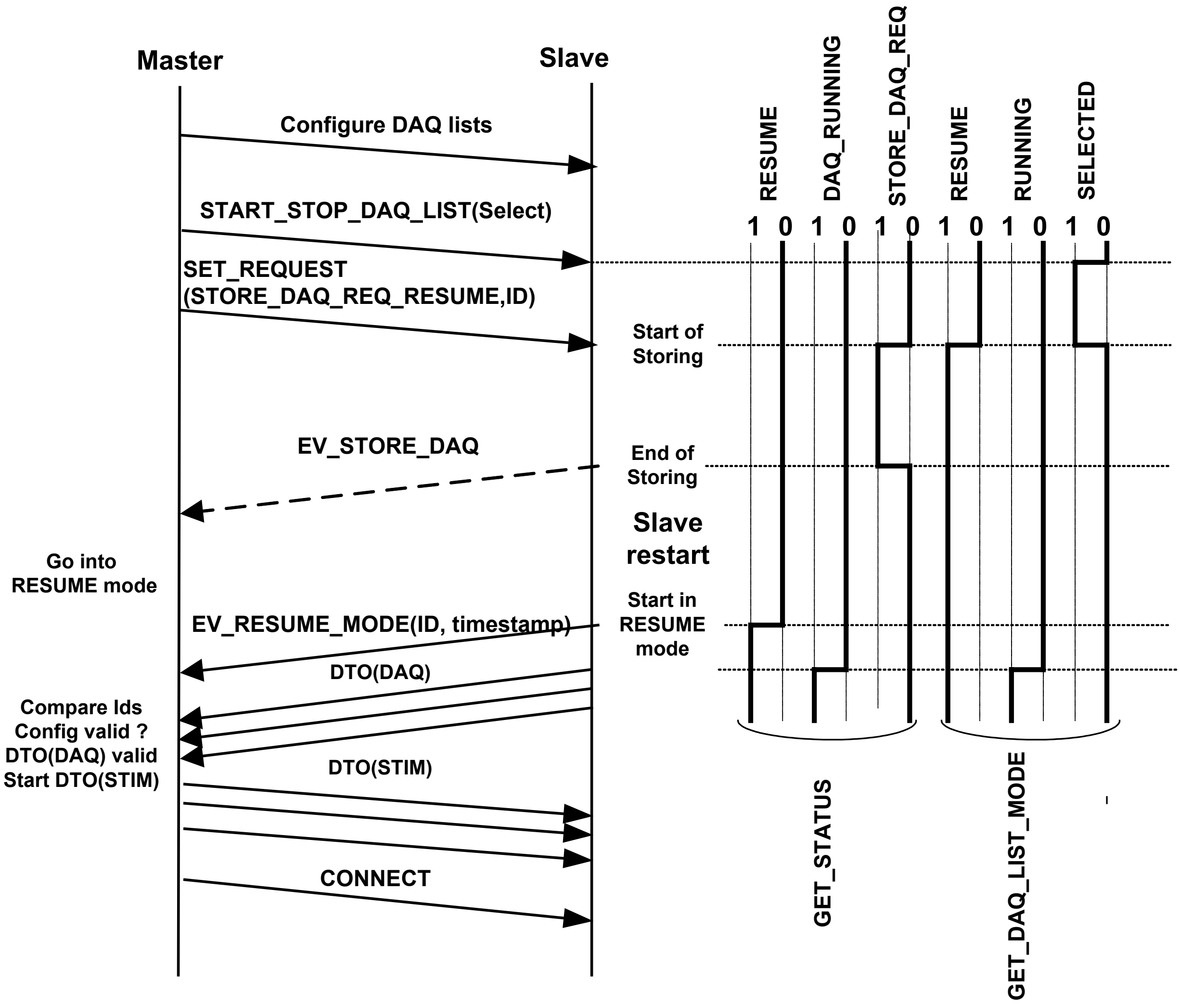

Figure 7 DAQ configuration storing with power-up data transfer (RESUME mode)

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

Table 4 Description of the RESUME mode event

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.

5.1.8 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.

5.1.9 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.

5.1.10 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. 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_ODT_TYPE_16: Type specific copy routines are used on ODT level. WORD (16 Bit) is the largest type, supported by the copy routines. GRANULARITY_ ODT_ENTRY_SIZE_DAQ and GRANULARITY_ODT_ENTRY_SIZE_STIM 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. MAX_ODT_ENTRY_SIZE_DAQ and MAX_ODT_ENTRY_SIZE_STIM 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. GRANULARITY_ ODT_ENTRY_SIZE_DAQ and GRANULARITY_ODT_ENTRY_SIZE_STIM 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. MAX_ODT_ENTRY_SIZE_DAQ and MAX_ODT_ENTRY_SIZE_STIM 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. GRANULARITY_ ODT_ENTRY_SIZE_DAQ and GRANULARITY_ODT_ENTRY_SIZE_STIM 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. MAX_ODT_ENTRY_SIZE_DAQ and MAX_ODT_ENTRY_SIZE_STIM 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 cannot 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.

5.1.11 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.

5.1.12 Synchronous Data Acquisition

By means of the DIRECTION flag, a DAQ list can be put in Synchronous Data Acquisition mode.

By means of DAQ with 0x00 <= PID <= 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-2 MC 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-2 MC description file, the slave can indicate that for this Event all data are sampled consistently.

If there is 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 is 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.

5.1.13 Synchronous Data Stimulation

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 <= PID <= 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.

5.2 Measurement Modes

5.2.1 Polling

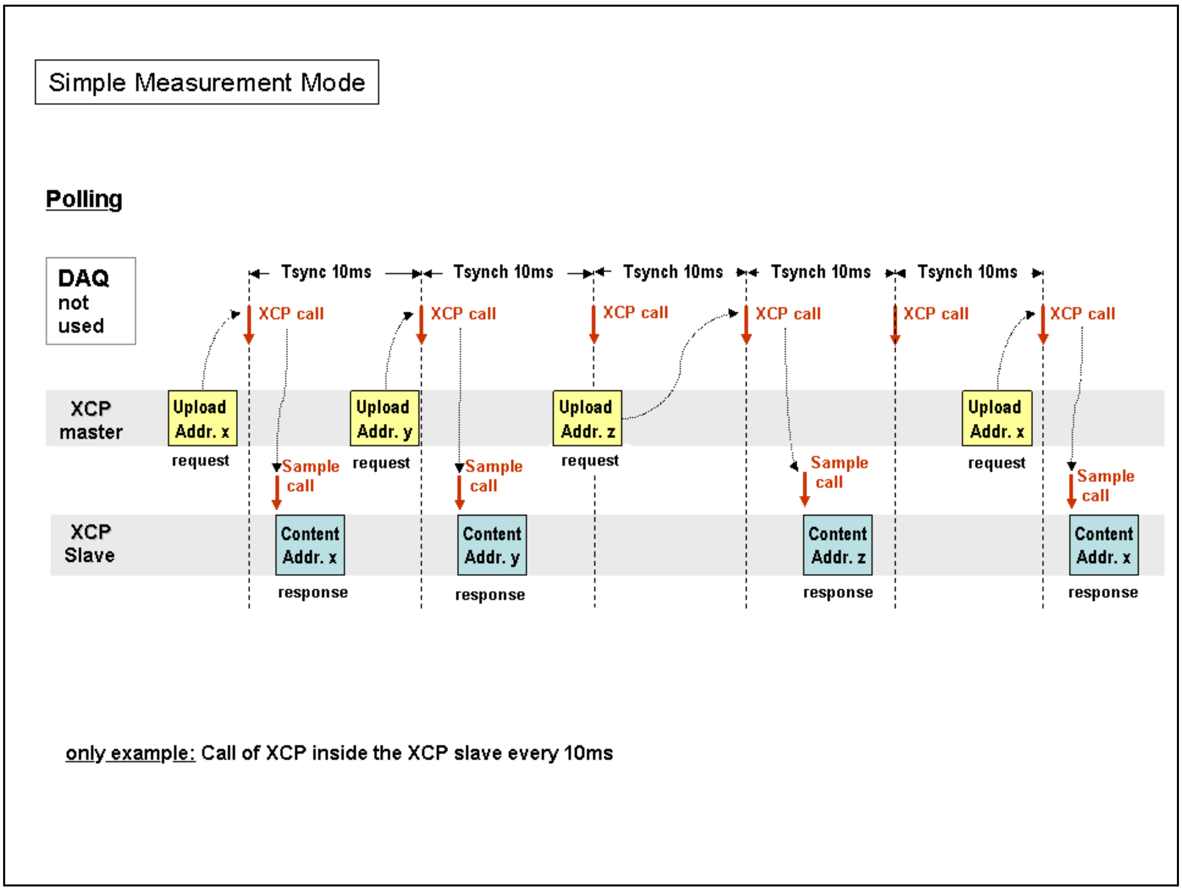

Figure 8 Measurement mode: polling

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 is 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`

5.2.2 Synchronous Data Transfer, DIRECTION=DAQ, Burst, Standard

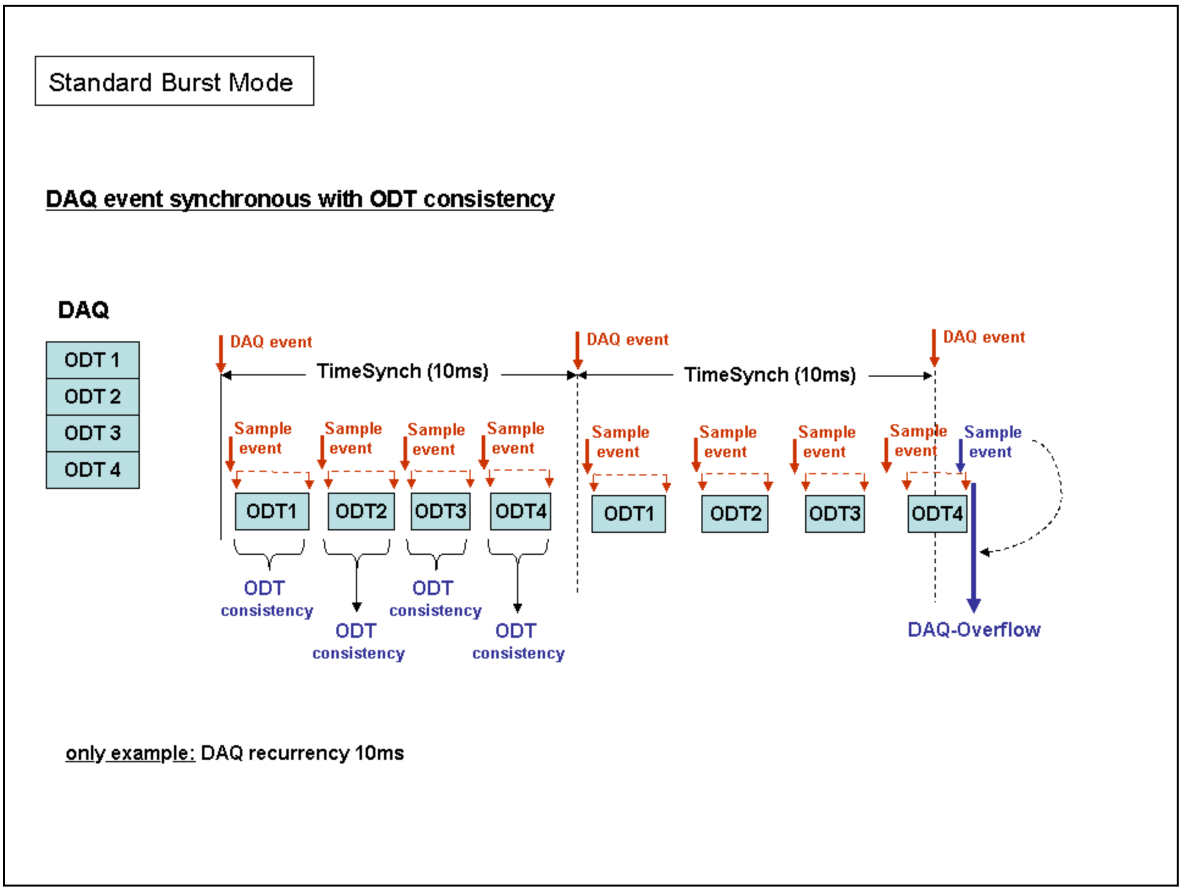

Figure 9 Measurement mode: standard burst

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`

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

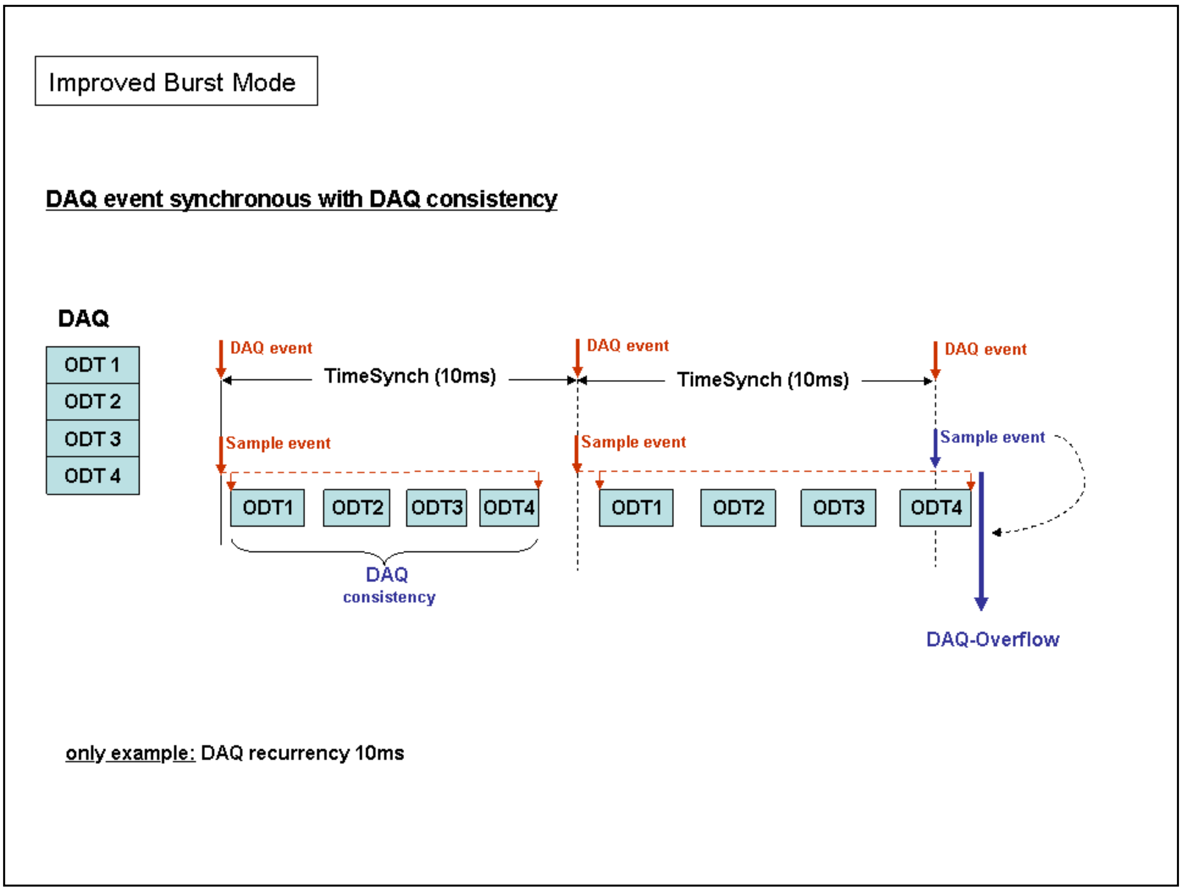

Figure 10 Measurement mode: improved burst

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-2 MC description file, the slave can indicate what kind of data consistency exists when data are processed within this Event.

5.2.4 Synchronous Data Transfer, DIRECTION=DAQ, Alternating

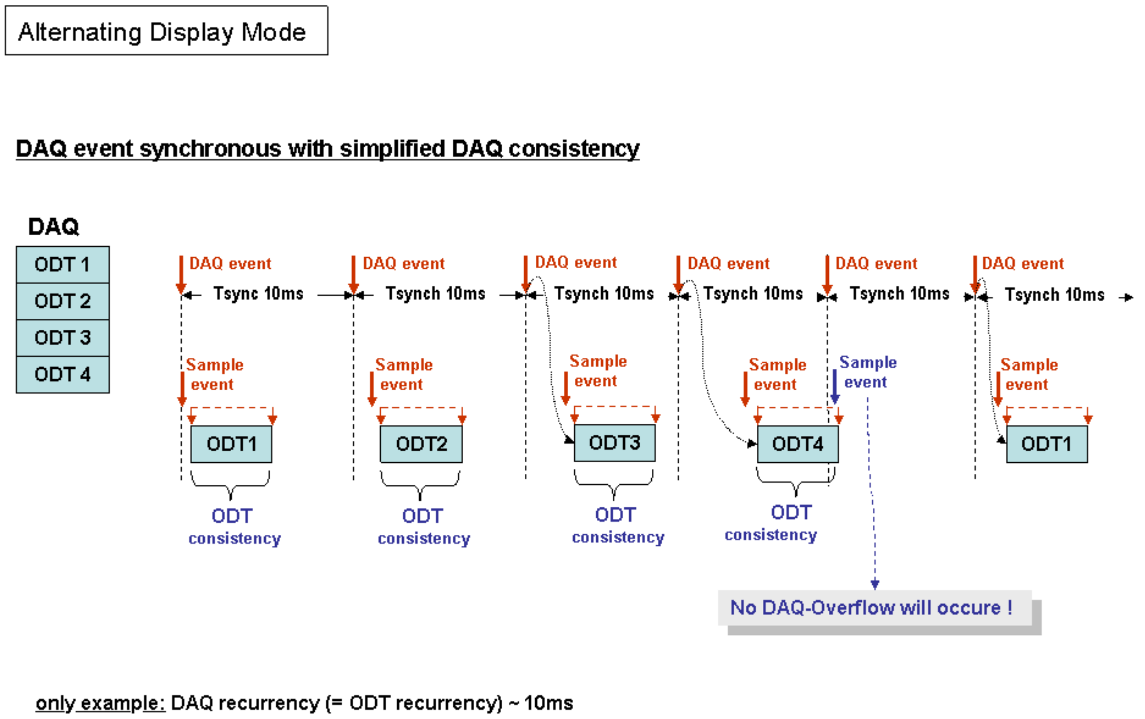
Alternating Display Mode
DAQ event synchronous with simplified DAQ consistency
DAQ
ODT 1
ODT 2
ODT 3
ODT 4
DAQ event
Tsync 10ms
Tsynch 10ms
Sample event
ODT1
ODT2
ODT3
ODT4
ODT consistency
No DAQ-Overflow will occure !
only example: DAQ recurrency (= ODT recurrency) ~ 10ms

Figure 11 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 is 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-2 MC description file is not allowed to use `TIMESTMAP_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.

5.3 Bypassing

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-2 MC 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 events 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.

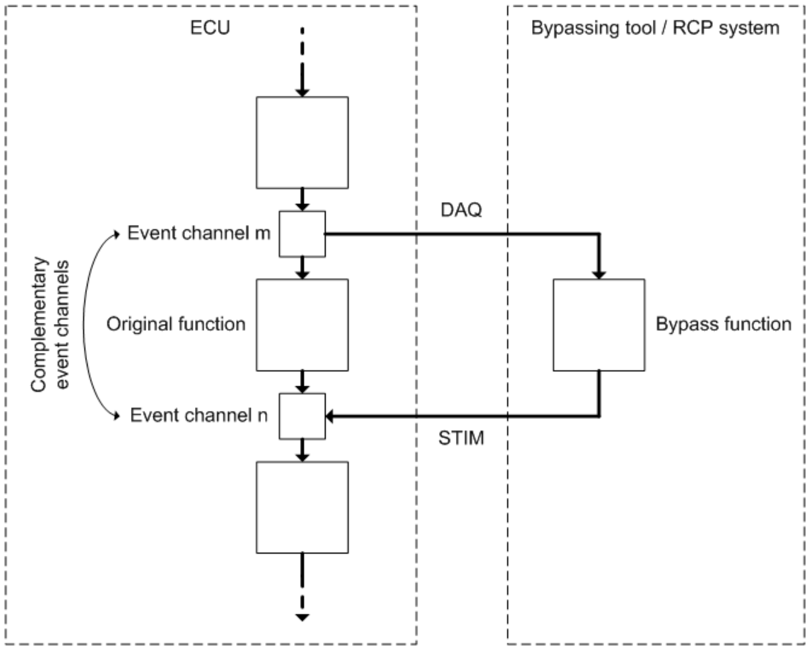

Figure 12 Bypassing

5.3.1 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.

5.3.2 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.

5.3.3 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.

5.3.4 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.

5.3.5 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.

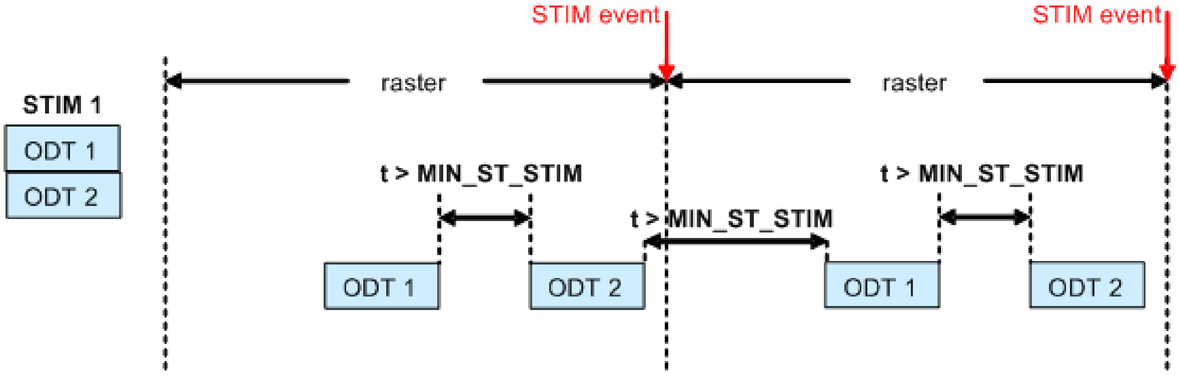

Figure 13 MIN_ST_STIM

5.4 Online Calibration

5.4.1 SECTOR, SEGMENT AND PAGE

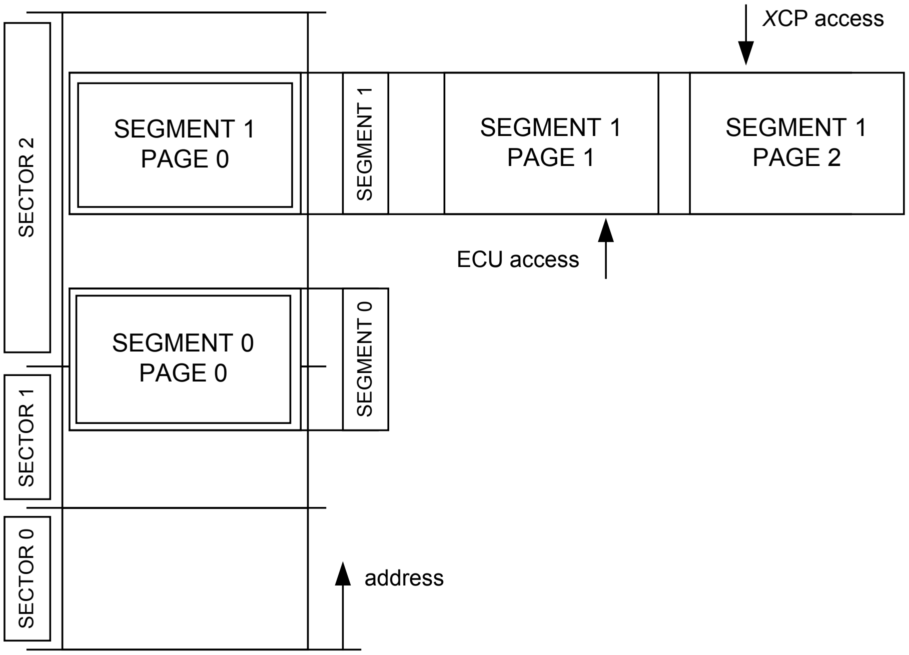

Figure 14 Calibration data organization

The slave’s memory layout 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 layout is described with objects called SECTORS.
SECTOR limits and sizes are important when reprogramming (Flashing) the slave device.

The logical layout 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 does not have to respect the limitations given by the start addresses and sizes of the SECTOR layout.

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.

5.4.2 Logical Layout: SEGMENT

The logical layout 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 does not have to respect the limitations given by the start addresses and sizes of the SECTOR layout (ref. Flashing).

A SEGMENT is described with the normal ASAM MCD-2 MC 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]

5.4.3 Accessibility - 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 does not 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 does not 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 does not 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]
```

5.4.4 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 cannot 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`.

5.4.5 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.

5.4.6 ADRESSING ACTION

The slave's memory layout 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.

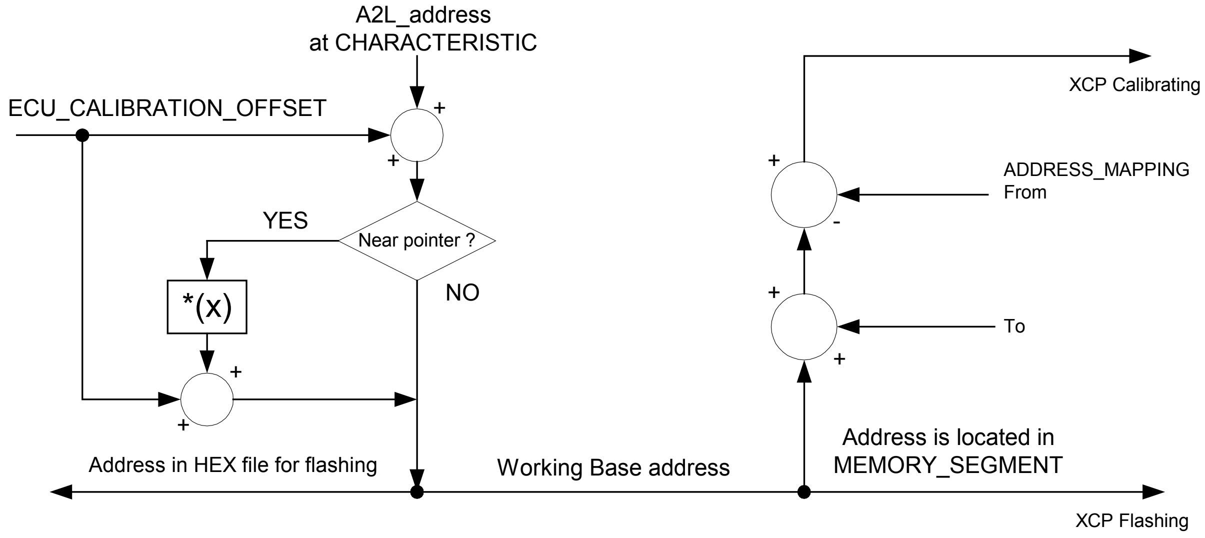

Figure 15 Address calculation

5.4.7 MASTER-SLAVE ACTION

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.

5.4.8 Page-Page Action

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.

5.5 Flash Programming

5.5.1 Physical Layout: SECTOR

The physical layout 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]
```

5.5.2 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 tools need identification information of the ECU and of the new flash content. XCP does not 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.

5.5.3 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.

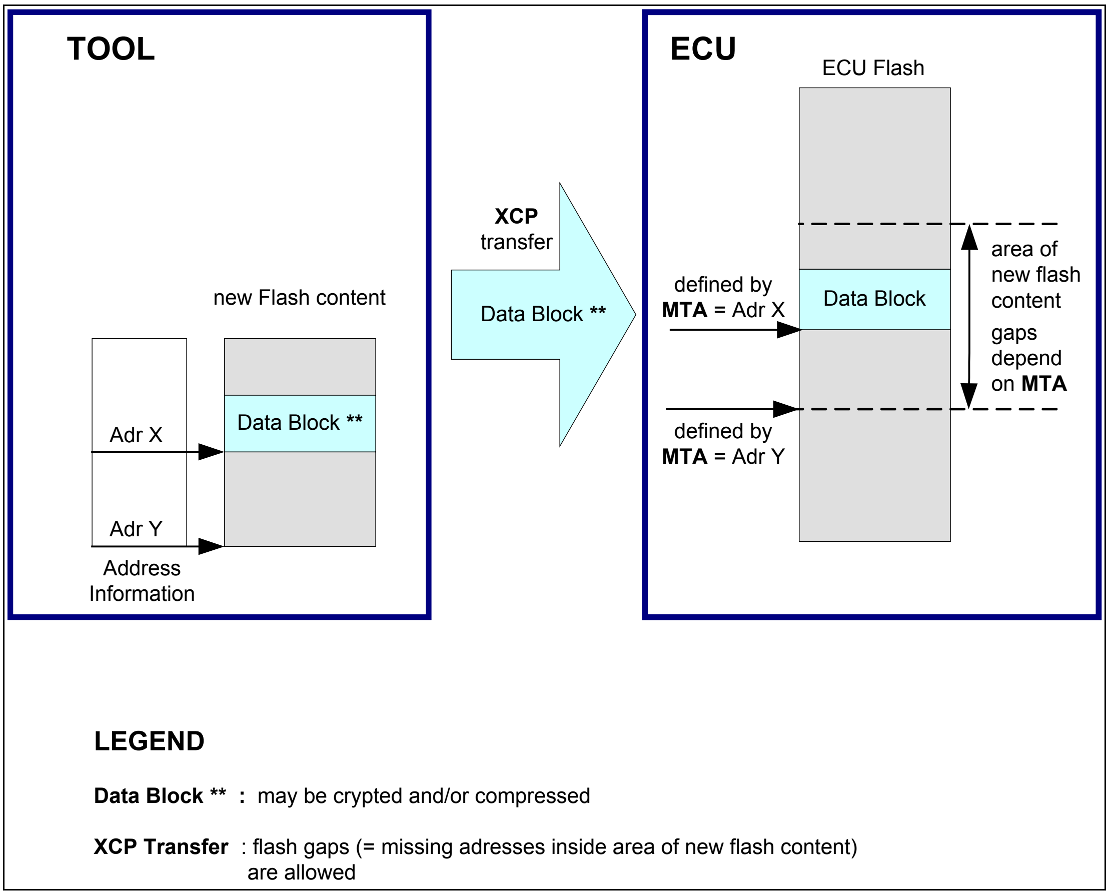

Figure 16 Absolute access mode

5.5.4 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.

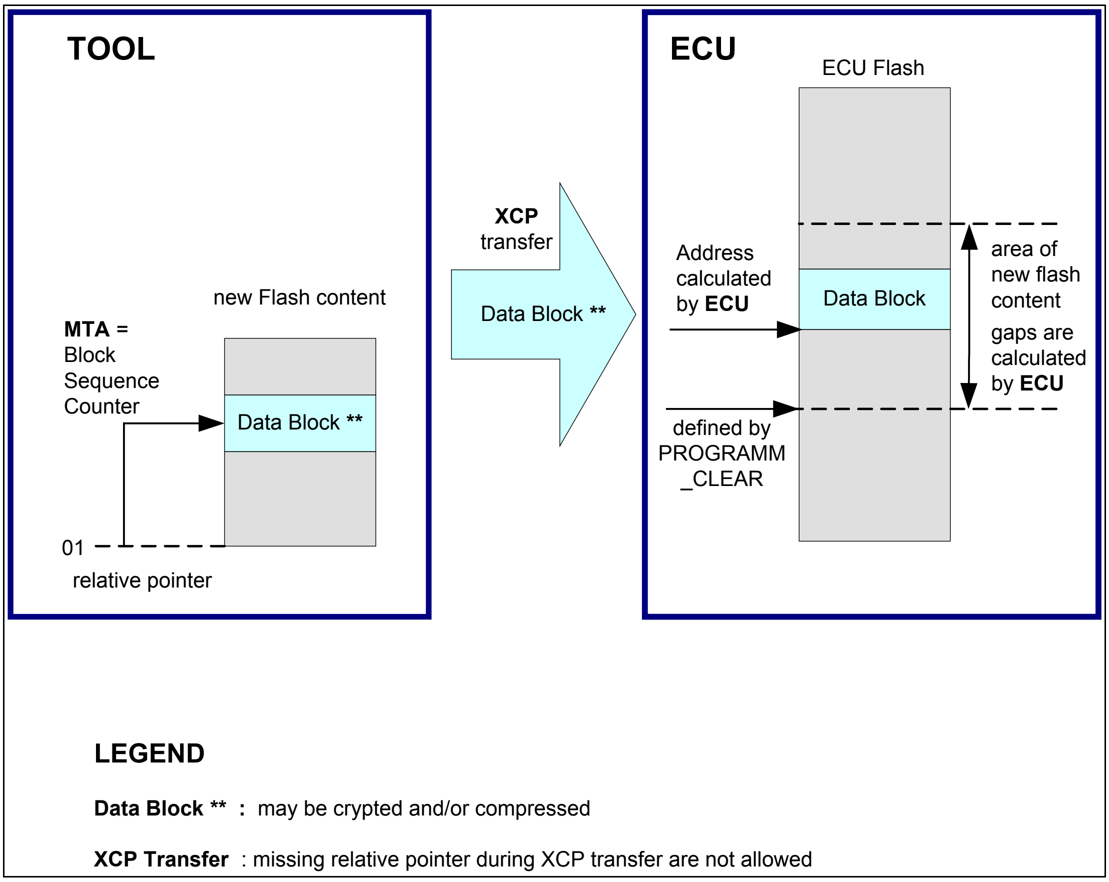

Figure 17 Functional access mode

The behaviour is similar to ISO 14229-1 [2] and ISO 15765-3 [3] .
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`

5.5.5 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
```

5.5.6 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
```

6 THE XCP PROTOCOL

6.1 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.

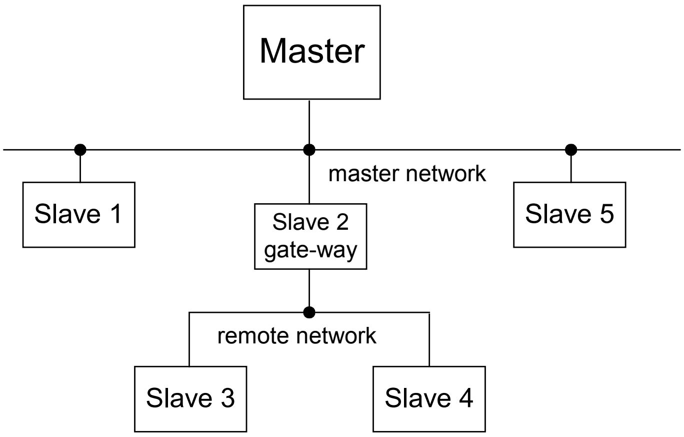

Figure 18 The XCP topology

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

<u>Master with Slave 1</u>
Master sends with CAN-Id = 0x100 on Master Network
Slave 1 sends with CAN-Id = 0x110 on Master Network

<u>Master with Slave 2 (Slave 2 directly)</u>
Master sends with CAN-Id = 0x200 on Master Network
Slave 2 sends with CAN-Id = 0x210 on Master Network

<u>Master with Slave 3 (Slave 2 as gate-way)</u>
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

6.2 The XCP Communication Models

6.2.1 The Standard Communication Model

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

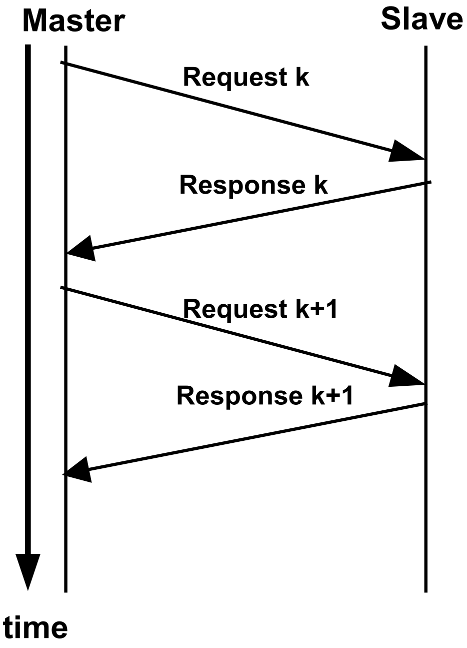

Figure 19 Standard communication model

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

6.2.2 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. [4]
The block transfer communication mode excludes interleaved communication mode.

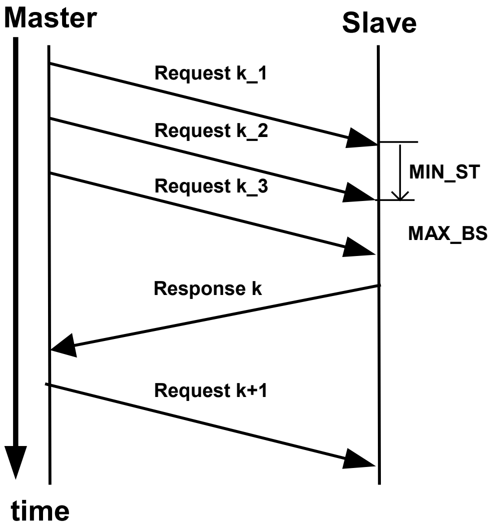

Figure 20 Master block transfer

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.

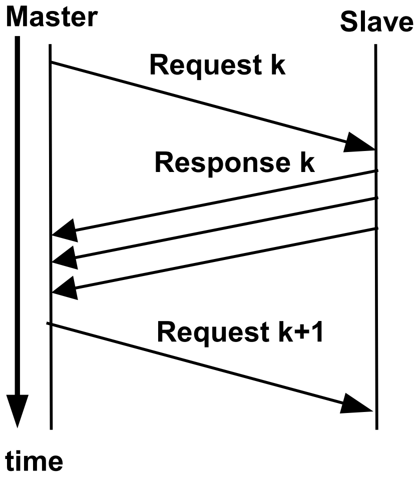

Figure 21 Slave block transfer

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.

6.2.3 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.

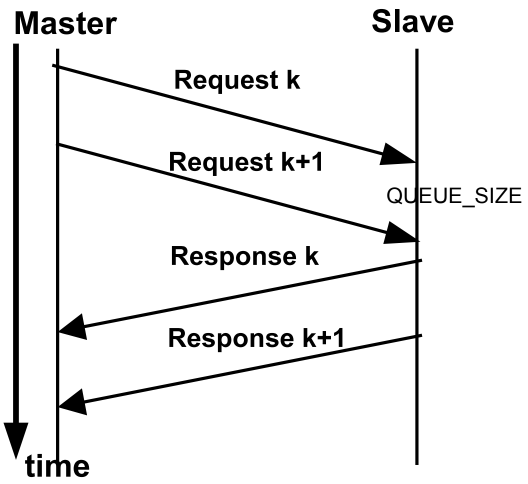

Figure 22 Interleaved communication model

`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.

6.3 STATE MACHINE

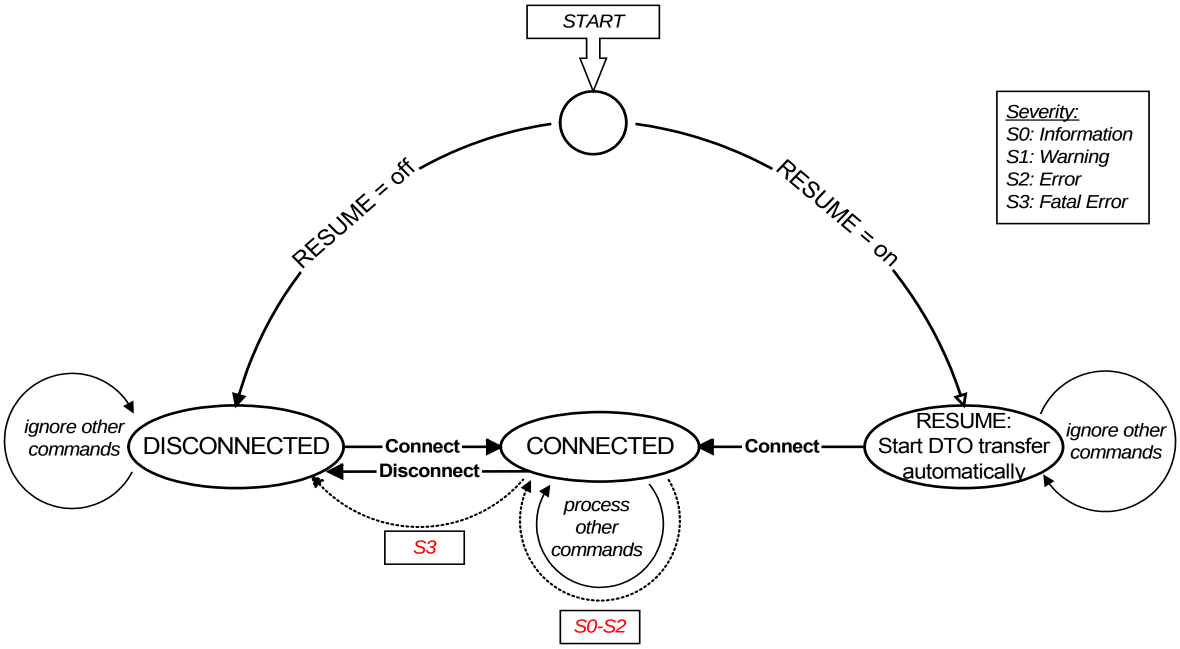

Figure 23 The XCP slave state machine

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

If there is no such a configuration available, the slave has to go to “DISCONNECTED” state.

In “DISCONNECTED” state, there is 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.

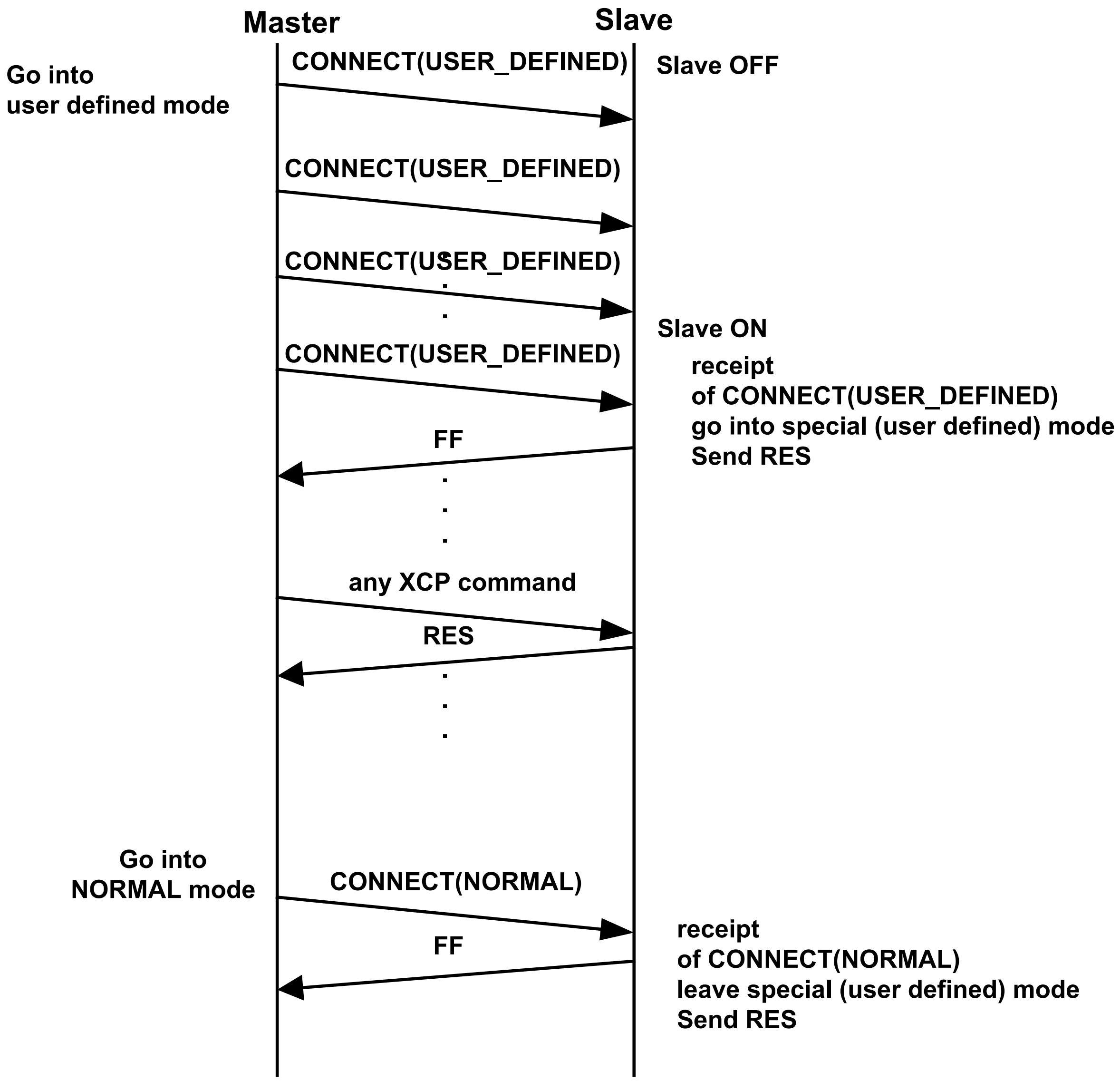

Figure 24 Typical use of CONNECT modes USER_DEFINED and NORMAL

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 is 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.

6.4 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.

6.5 The XCP Message (Frame) Format

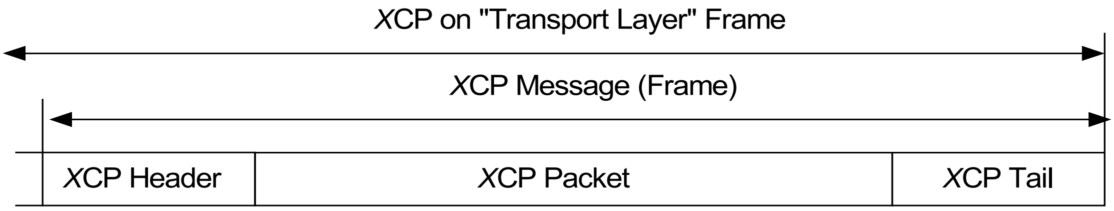

Figure 25 The XCP message (frame) format

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.
Chapter 8.1 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.
The content of the Control Fields is described in the associated standard of the respective transport layer ([6] [7] [8] [9] [10]).

7 THE LIMITS OF PERFORMANCE

7.1 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.

Table 5 Overview of generic DAQ performance parameters

Name	Type	Representation	Range of value
MAX_CTO	Parameter	BYTE	0x08 – 0xFF
MAX_DTO	Parameter	WORD	0x0008 – 0xFFFF
MAX_DTO_STIM	Parameter	WORD	0x0008 – 0xFFFF

The range of these protocol parameters can be smaller, depending on the used transport layer.
If MAX_DTO_STIM is defined, MAX_DTO applies only for DTOs having direction DAQ .
If MAX_DTO_STIM is not defined, MAX_DTO applies for both directions.

7.2 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.

Table 6 Overview of EVENT specific performance parameters

Name	Type	Represen-tation	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.

Table 7 Overview of DAQ list specific performance parameters

Name	Type	Representation	Range of value
MAX_DAQ	Parameter	WORD	0x0000 – 0xFFFF
MAX_DAQ_TOTAL	Constant	WORD	0x0000 – 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.

Table 8 Overview of ODT specific performance parameters

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

7.2.1 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.

Table 9 ODT parameters of a specific DAQ list of direction DAQ

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

7.2.2 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.

Table 10 ODT parameters of a DAQ list of direction STIM

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

7.2.3 ECU Resource Consumptions

This section covers the aspect of calculating the ECU resource consumption caused by DAQ/STIM measurement configuration. These resources are ECU RAM consumption and CPU execution time. The measurement configuration is basically a list of measurement variables and their corresponding XCP events.

In order to calculate the specific resource consumption, a set of mathematical formulas is defined. These have parameters which are specific to an XCP protocol implementation of an ECU.

Furthermore, parameters for the limits of these resources are defined.
A calibration tool can use this information to inform the calibration engineer, particularly if the defined limits are exceeded, to avoid e.g. physical damage of the controlled device.

7.2.3.1 ECU RAM Consumption

The DAQ processor of the XCP slave stores the measurement configuration in the RAM of the ECU.

The RAM consumption for the XCP DAQ measurement configuration is calculated by the following formulas.

$$Total\ DAQ\ Memory\ Consumption = \sum_{i}^{Events} RAM\big(Event(i)\big)$$

$$RAM\big(Event(i)\big) = \sum_{j}^{DAQ\ Lists\ (Event(i))} RAM\big(DAQList(j)\big)$$

$$RAM\left(DAQList_{DAQ}(j)\right) = DAQ\_SIZE + \sum_{k}^{ODTs\ \left(DAQList_{DAQ}(j)\right)} RAM\ \left(ODT_{DAQ}(k)\right)$$

$$RAM(DAQList_{STIM}(j)) = \text{DAQ\_SIZE} + \sum_{k}^{ODTs\,(DAQList_{STIM}(j))} RAM\left(ODT_{STIM}(k)\right)$$

$$RAM\,(ODT_{DAQ}(k)) = \text{ODT\_SIZE}$$

$$+\,\text{ODT\_ENTRY\_SIZE} * \sum_{l}^{ODT\ entries\,\left(ODT_{DAQ}(k)\right)} 1$$

$$+\,ODTPayload(k) * \text{ODT\_DAQ\_BUFFER\_ELEMENT\_SIZE}$$

$$RAM\,(ODT_{STIM}(k)) = \text{ODT\_SIZE}$$

$$+\,\text{ODT\_ENTRY\_SIZE} * \sum_{l}^{ODT\ entries\,\left(ODT_{STIM}(k)\right)} 1$$

$$+\,ODTPayload(k) * \text{ODT\_STIM\_BUFFER\_ELEMENT\_SIZE}$$

ODTPayload(k), the ODT payload size for a given ODT(k), has to be calculated by the XCP master and is determined by the measurement signal configuration. The same applies for the limits of the sums over events, DAQ lists and ODTs.

Table 11 Parameters for the calculation of RAM consumption of an XCP DAQ measurement configuration (definition located at IF_DATA XCP)

Name	Representation	Description
ODT_SIZE	uint	Number of memory elements needed for storage of one ODT configuration
ODT_ENTRY_SIZE	uint	Number of memory elements needed for storage of one ODT entry
ODT_DAQ_BUFFER_ELEMENT_SIZE	uint	Size of memory elements to be reserved in the send queue, direction DAQ. The parameter may be 0 for the case that the XCP slave does not buffer the data for transmission in RAM.
ODT_STIM_BUFFER_ELEMENT_SIZE	uint	Size of memory elements to be reserved in the receive queue, direction STIM

Name	Representation	Description
DAQ_SIZE	uint	Number of memory elements needed for storage of one DAQ list configuration
DAQ_MEMORY_LIMIT	ulong	The total size of available DAQ configuration memory

All element sizes and factors are multiples of the address granular factor AG, e.g. for AG = 1, a memory element is one byte.

7.2.3.2 CPU Execution Time

The XCP data acquisition inside an ECU normally causes CPU load, because the configured measurement data must be transferred from its original memory locations to a send queue and transmitted by the transport layer and lower layers.
The resulting CPU load can be approximated by the following formulas. They do not claim to model every possible implementation exactly, but to yield a result which is a good estimation for the generated CPU load and can ensure that the measurement configuration does not violate the limits in order to avoid physical damage to the controlled unit.

$$Total\ CPU\ Load\ = \sum_{i}^{Events} CPULoad(Event(i))$$

$$CPULoad(Event(i)) = \frac{\sum_{j}^{DAQ\ Lists\ (Event(i))} CPULoad(DAQList(j))}{CycleTime(Event(i))[s]}$$

In the case, that the event is not periodic, CycleTime must be replaced with the minimal cycle time specified by the IF_DATA block "MIN_CYCLE_TIME".

$$CPULoad(DAQList_{DAQ/STIM}(j)\ ) = DAQ\_FACTOR$$

$$+ \sum_{k}^{ODTs\ (DAQList(j))} \left(CPULoad(ODT\_QUEUE(k) + CPULoad\ (ODT(k))\right)$$

$$CPULoad\ (ODT\_QUEUE(k)) = \mathrm{ODT\_FACTOR}_{\mathrm{Queue}}$$

$$+ \mathrm{ODT\_ELEMENT\_LOAD} * ODTPayload(k)$$

$$CPULoad(ODT_{DAQ/STIM}(k)) = ODT\_FACTOR$$

$$+ ODT\_ENTRY\_FACTOR \ * \sum^{ODT\ entries(ODT(k))} 1$$

$$+ \sum_{n}^{OESFT} (SIZE\_FACTOR[n]) * \sum^{ODTentries(ODT(k),size=SIZE[n])} SIZE[n]$$

The abbreviation OESFT is short for ODT entry size factor table. The sum over this table iterates over all table entries and sums up all ODT entries with the corresponding size.
A more detailed explanation follows below the next table.

Table 12 Parameters for the CPU load calculation of an XCP DAQ measurement configuration (definition located at IF_DATA)

Name	Representation	Description
DAQ_FACTOR	float	Basic CPU load to be considered for each DAQ list. Part of CPU_LOAD_CONSUMPTION_DAQ CPU_LOAD_CONSUMPTION_STIM
ODT_FACTOR	float	Basic CPU load to be considered for processing each ODT. Part of CPU_LOAD_CONSUMPTION_DAQ CPU_LOAD_CONSUMPTION_STIM
ODT_FACTOR_{Queue}	float	Basic CPU load to be considered for buffering each ODT into the transmission queue. Part of CPU_LOAD_CONSUMPTION_QUEUE
ODT_ELEMENT_LOAD	float	CPU load caused by copying one single element
ODT_ENTRY_FACTOR	float	CPU load caused by the handling of one single ODT entry
SIZE[n]	uint	Part of ODT_ENTRY_SIZE_FACTOR_TABLE of blocks CPU_LOAD_CONSUMPTION_DAQ CPU_LOAD_CONSUMPTION_STIM
SIZE_FACTOR[n]	float	Part of ODT_ENTRY_SIZE_FACTOR_TABLE of blocks

Name	Representation	Description
DAQ_FACTOR	float	Basic CPU load to be considered for each DAQ list. Part of CPU_LOAD_CONSUMPTION_DAQ CPU_LOAD_CONSUMPTION_STIM
ODT_FACTOR	float	Basic CPU load to be considered for processing each ODT. Part of CPU_LOAD_CONSUMPTION_DAQ CPU_LOAD_CONSUMPTION_STIM
		CPU_LOAD_CONSUMPTION_DAQ CPU_LOAD_CONSUMPTION_STIM
CPU_LOAD_MAX_TOTAL	float	Total CPU load limit regarding the DAQ measurement, part of IF_DATA block "DAQ"
$\mathrm{CPU\_LOAD\_MAX}_{\mathrm{Event}}$	float	CPU load limit for one single event, part of IF_DATA block "event"

The XCP master can use the calculated results to report the percentage of CPU load with regard to the limits CPU_LOAD_MAX_TOTAL resp. CPU_LOAD_MAX on the event level. This is the reason why no CPU load unit is defined, because for the percentage, a unit which applies for both numerator and denominator is reduced.

The IF_DATA block CPU_LOAD_CONSUMPTION_DAQ describes the load consumption for the memory copy routine. If this is defined, the table ODT_ENTRY_SIZE_FACTOR_TABLE must be defined mandatorily and must contain at least one record. Each record consists of a size and a corresponding load factor which is applied to all ODT entries having the specific size or a multiple of it.
If an ODT entry has a size which is not described by any of the table records, the next smaller size entry shall be applied. If no smaller size is defined, the next larger size shall be applied.
The selected size must be considered multiple times until the result is greater than or equal to the size of the ODT entry.

Example:

```
/begin CPU_LOAD_CONSUMPTION_DAQ
     1     // "DAQ_FACTOR"
     2     // "ODT_FACTOR"
     3     // "ODT_ENTRY_FACTOR"
       /begin ODT_ENTRY_SIZE_FACTOR_TABLE
          1     //"SIZE"
          150   // "SIZE_FACTOR", e.g. CPU cycles
       /end ODT_ENTRY_SIZE_FACTOR_TABLE
       /begin ODT_ENTRY_SIZE_FACTOR_TABLE
          4     //"SIZE"
          420   // "SIZE_FACTOR"
```

```
                    /end ODT_ENTRY_SIZE_FACTOR_TABLE
             /end CPU_LOAD_CONSUMPTION_DAQ
```

If an ODT entry has the size 13, i.e. **3** * 4 + 1, the resulting load for the ODT entry is **4** * **420** = 1680.
Note that additional load is generated from the containing ODT and DAQ list.
More examples are available in Table 225.

8 THE XCP PROTOCOL LAYER

8.1 THE XCP PACKET

8.1.1 THE XCP PACKET TYPES

All XCP communication is transferred as data objects called XCP Packets.

There are 2 basic Packet types:

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

The CTO (Command Transfer Object) 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 (Data Transfer Object) is used for transmitting synchronous data acquisition data (DAQ) and for transmitting synchronous data stimulation data (STIM).

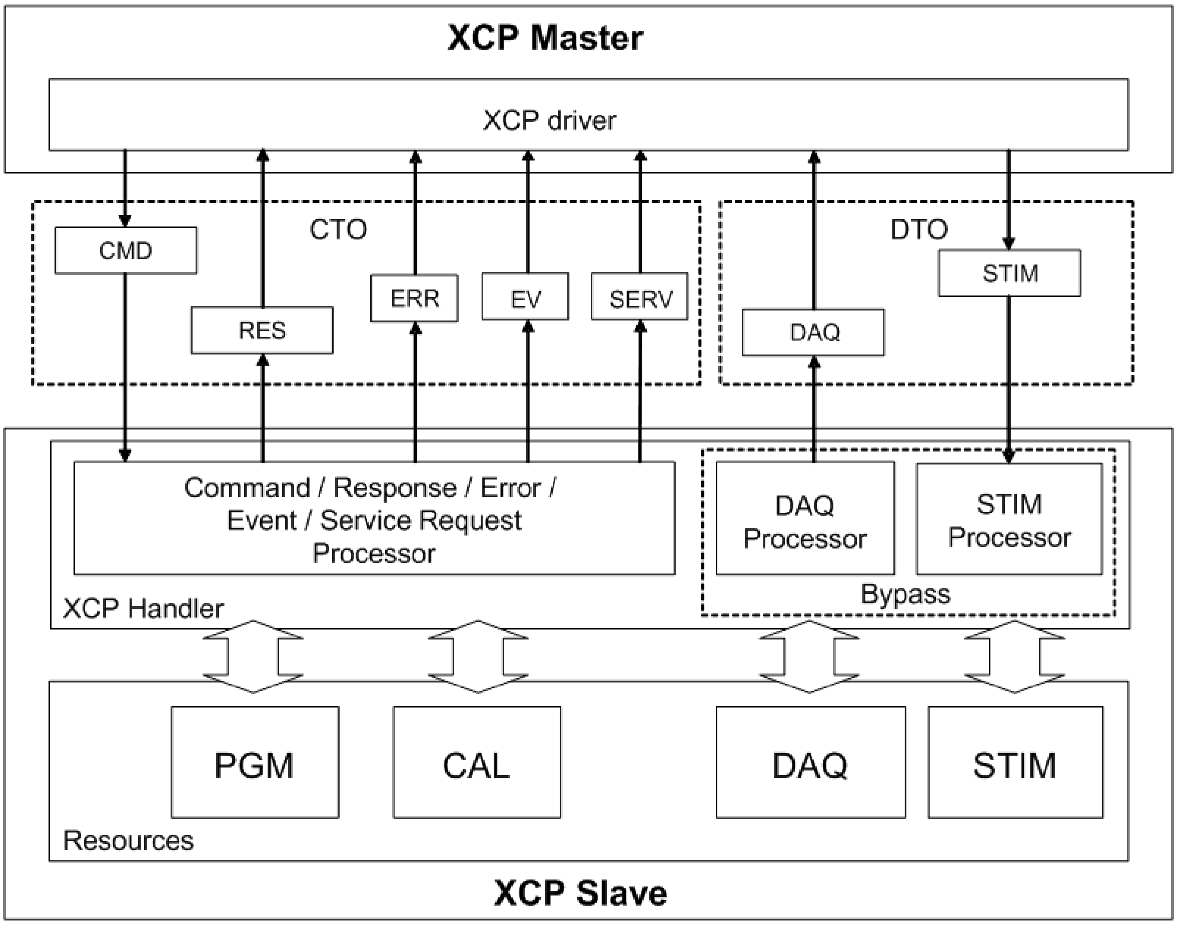

Figure 26 Communication flow between master and slave devices

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.

The XCP Handler may not always have access to the resources of the XCP slave.
With `ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE` the XCP Handler can indicate this situation to the master.

8.1.2 THE XCP PACKET FORMAT

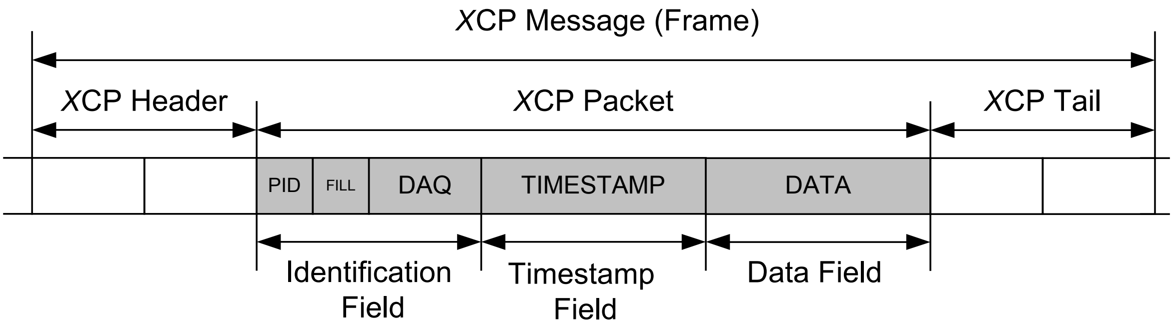

Figure 27 The XCP packet format

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.

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

8.1.2.1 THE IDENTIFICATION FIELD

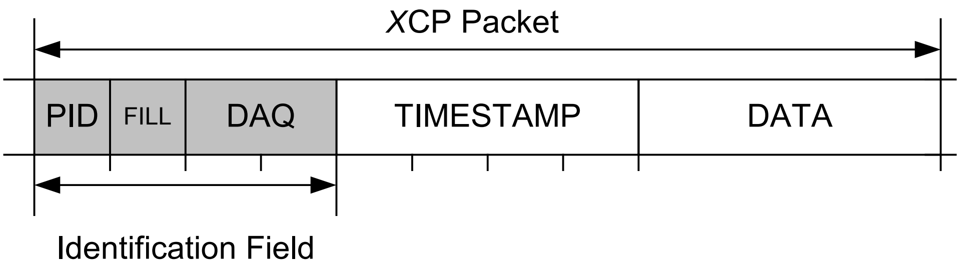

Figure 28 The XCP packet identification field

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

For this purpose, an XCP Packet basically always starts with an Identification Field which as first byte contains the Packet IDentifier (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.

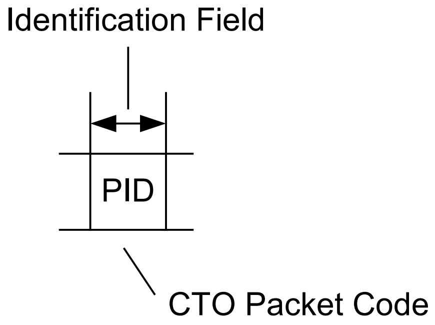

Figure 29 Identification field type "CTO packet code"

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:

Table 13 Relative ODT numbering for DAQ lists

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. All following ODTs of a DAQ list transmission cycle need not be in ascending order but of course complete.
`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_PID`, the slave has to make sure that for every ODT there is 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.

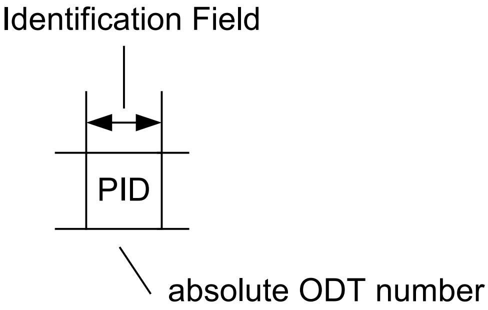

Figure 30 Identification field type "absolute ODT number"

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.

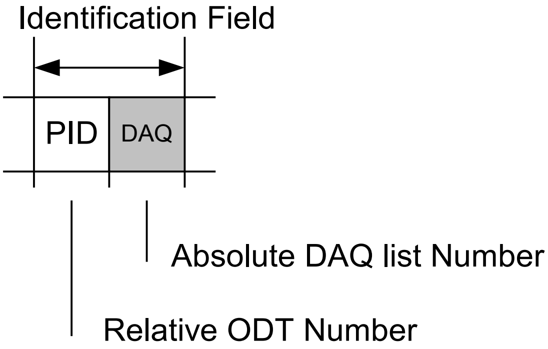

Figure 31 Identification field type "relative ODT number and absolute DAQ list number (BYTE)"

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

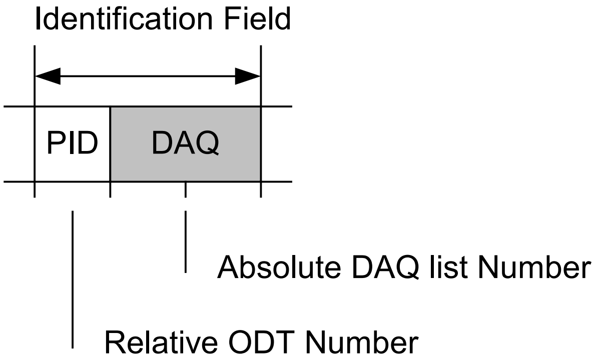

Figure 32 **Identification field type "relative ODT number and absolute DAQ list number (WORD)"**

If for the XCP Packet certain alignment conditions have to be met, there is the possibility to transfer an extra FILL byte.

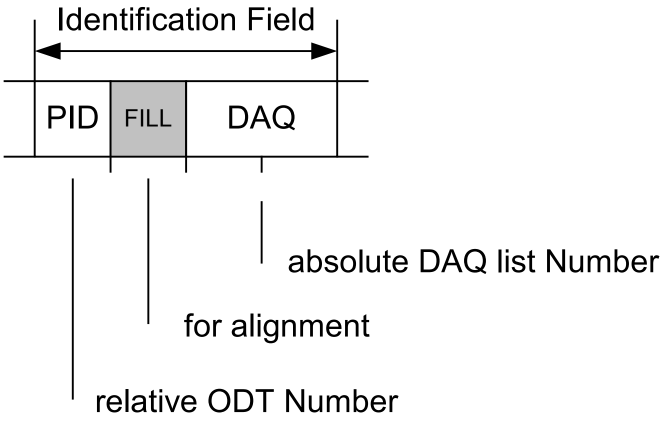

Figure 33 **Identification field type "relative ODT number and absolute DAQ list number (WORD, aligned)"**

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 XCP 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.

8.1.2.2 THE TIMESTAMP FIELD

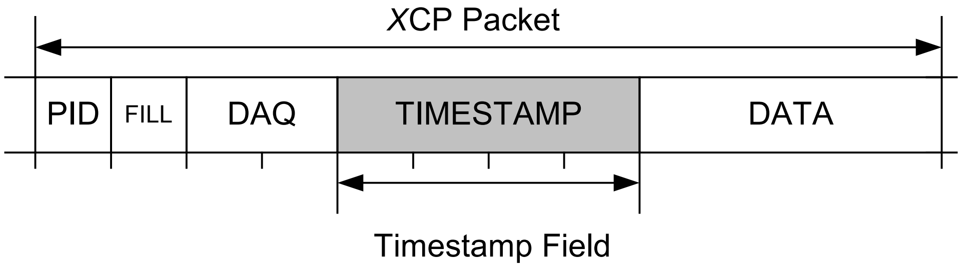

Figure 34 The XCP packet timestamp field

An XCP 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 TimeStamp (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 cannot switch off the time stamp with `SET_DAQ_LIST_MODE`.

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.

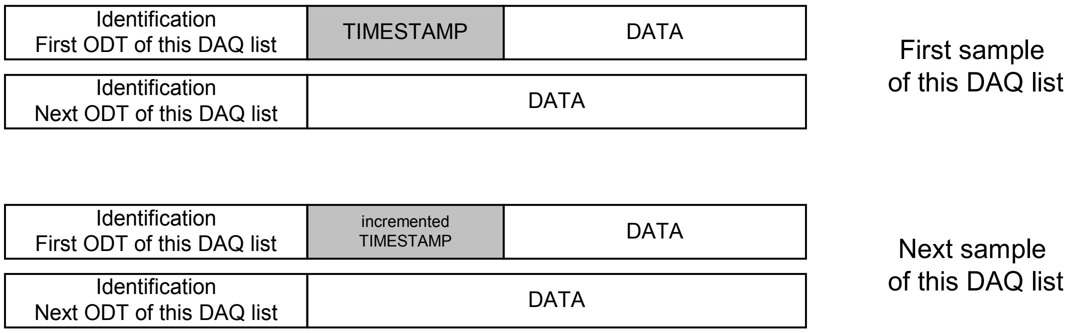

Figure 35 TS only in first DTO packet of sample

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

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.

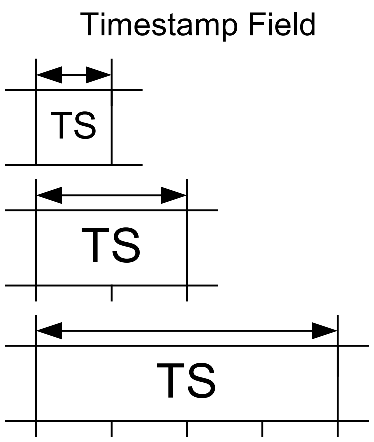

Figure 36 Timestamp field types

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.

8.1.2.3 THE DATA FIELD

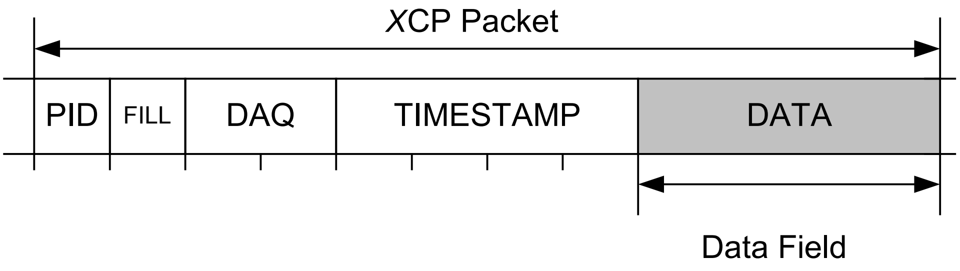

Figure 37 The XCP packet data field

An XCP 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.

8.1.3 THE CTO PACKETS

The CTO is used for transferring generic control commands.

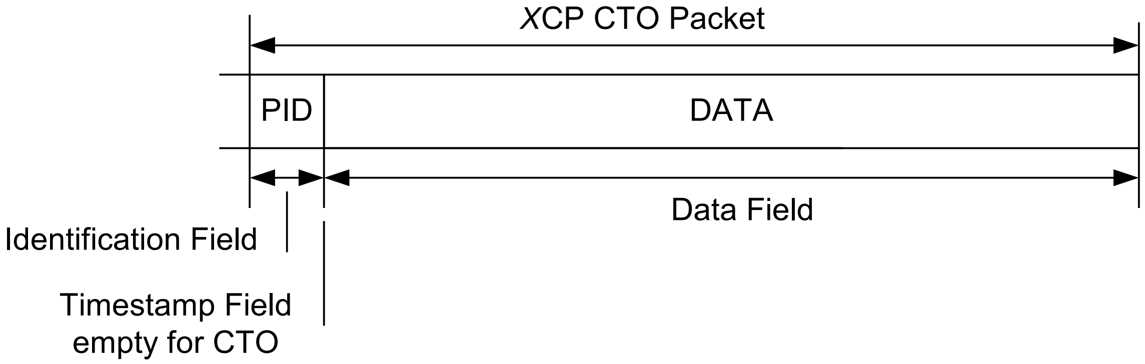

Figure 38 The CTO packet

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.

8.1.3.1 COMMAND PACKET

Table 14 Command packet structure

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

The PID contains the ComManD 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.

8.1.3.2 COMMAND RESPONSE PACKET

Table 15 Command response packet structure

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

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

8.1.3.3 ERROR PACKET

Table 16 Error packet structure

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 document.
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.

8.1.3.4 EVENT PACKET

Table 17 Event packet structure

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.
All possible event codes are defined in the section "Table of Event Codes (EV)" in this paper. The structure of all possible events is defined in the "Description of Events" section of 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.

8.1.3.5 SERVICE REQUEST PACKET

Table 18 Service request packet structure

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.

8.1.4 THE DTO PACKETS

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

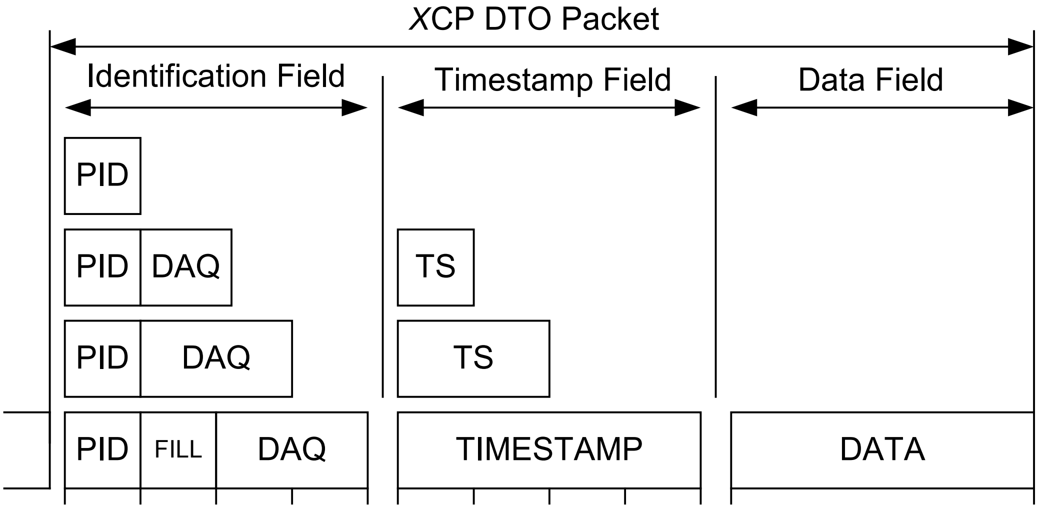

Figure 39 The DTO packet

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.

8.1.4.1 DATA ACQUISITION PACKET

Table 19 Data acquisition packet structure

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 Object Descriptor Table (ODT) that describes which data acquisition elements are contained in the remaining data bytes.

8.1.4.2 SYNCHRONOUS DATA STIMULATION PACKET

Table 20 Synchronous data stimulation packet structure

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 Object Descriptor Table (ODT) that describes which data stimulation elements are contained in the remaining data bytes.

8.1.5 The XCP 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.

8.1.5.1 Master -> Slave

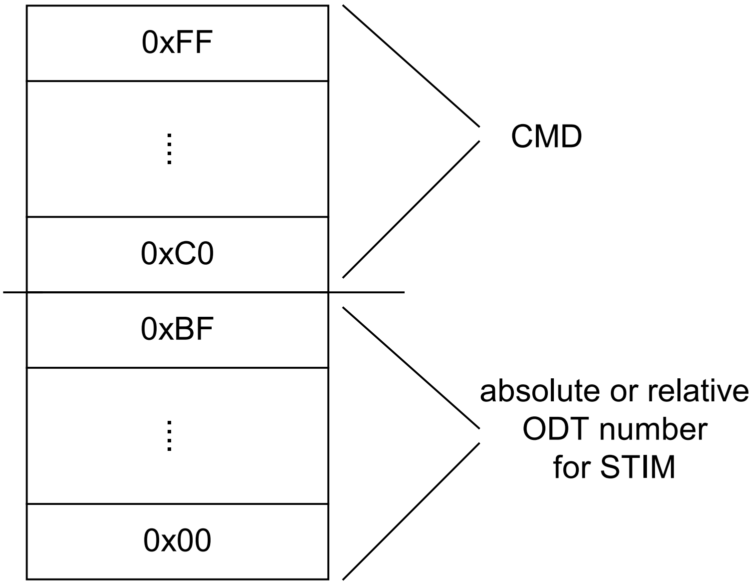

Figure 40 The XCP packet IDentifiers from master to slave

8.1.5.2 Slave -> Master

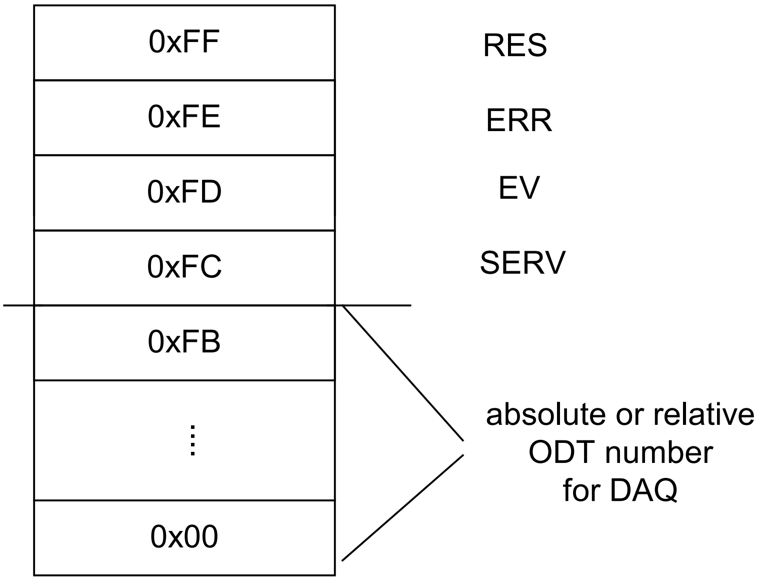

Figure 41 The XCP packet IDentifiers from slave to master

8.2 Event Codes

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.

Table 21 Event code overview

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_TIME_SYNC	0x08	Transfer of externally triggered timestamp	S0
EV_STIM_TIMEOUT	0x09	Indication of a STIM timeout	S0-S3
EV_SLEEP	0x0A	Slave entering SLEEP mode	S1
EV_WAKE_UP	0x0B	Slave leaving SLEEP mode	S1
EV_USER	0xFE	User-defined event	S0
EV_TRANSPORT	0xFF	Transport layer specific event	See associated standards [6] [7] [8] [9] [10]

8.3 Service Request Codes

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.

Table 22 Service request codes

Service Request	Code	Description
SERV_RESET	0x00	Slave requesting to be reset
SERV_TEXT	0x01	Slave transferring a byte stream of plain ASCII text.
		The line separator is LF or CR/LF.
		The text can be transferred in consecutive packets.
		The end of the overall text is indicated by the last packet containing a Null terminated string.

8.4 Command Codes

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.

Table 23 Standard commands

Command	Code	Support
CONNECT	0xFF	mandatory
DISCONNECT	0xFE	mandatory
GET_STATUS	0xFD	mandatory
SYNCH	0xFC	mandatory
GET_COMM_MODE_INFO	0xFB	optional
GET_ID	0xFA	optional
SET_REQUEST	0xF9	optional
GET_SEED	0xF8	optional
UNLOCK	0xF7	optional
SET_MTA	0xF6	optional
UPLOAD	0xF5	optional
SHORT_UPLOAD	0xF4	optional
BUILD_CHECKSUM	0xF3	optional
TRANSPORT_LAYER_CMD	0xF2	optional
USER_CMD	0xF1	optional

Table 24 Calibration commands

Command	Code	Support
DOWNLOAD	0xF0	mandatory
DOWNLOAD_NEXT	0xEF	optional
DOWNLOAD_MAX	0xEE	optional
SHORT_DOWNLOAD	0xED	optional
MODIFY_BITS	0xEC	optional

Table 25 Page switching commands

Command	Code	Support
SET_CAL_PAGE	0xEB	optional
GET_CAL_PAGE	0xEA	optional
GET_PAG_PROCESSOR_INFO	0xE9	optional
GET_SEGMENT_INFO	0xE8	optional
GET_PAGE_INFO	0xE7	optional
SET_SEGMENT_MODE	0xE6	optional
GET_SEGMENT_MODE	0xE5	optional
COPY_CAL_PAGE	0xE4	optional

Table 26 Basic data acquisition and stimulation commands

Command	Code	Support
SET_DAQ_PTR	0xE2	mandatory
WRITE_DAQ	0xE1	mandatory
SET_DAQ_LIST_MODE	0xE0	mandatory
START_STOP_DAQ_LIST	0xDE	mandatory
START_STOP_SYNCH	0xDD	mandatory
WRITE_DAQ_MULTIPLE	0xC7	optional
READ_DAQ	0xDB	optional
GET_DAQ_CLOCK	0xDC	optional
GET_DAQ_PROCESSOR_INFO	0xDA	optional
GET_DAQ_RESOLUTION_INFO	0xD9	optional
GET_DAQ_LIST_MODE	0xDF	optional
GET_DAQ_EVENT_INFO	0xD7	optional

Table 27 Static data acquisition and stimulation commands

Command	Code	Support
CLEAR_DAQ_LIST	0xE3	mandatory
GET_DAQ_LIST_INFO	0xD8	optional

Table 28 Dynamic data acquisition and stimulation commands

Command	Code	Support
FREE_DAQ	0xD6	mandatory
ALLOC_DAQ	0xD5	mandatory
ALLOC_ODT	0xD4	mandatory
ALLOC_ODT_ENTRY	0xD3	mandatory

Table 29 Non-volatile memory programming commands

Command	Code	Support
PROGRAM_START	0xD2	mandatory
PROGRAM_CLEAR	0xD1	mandatory
PROGRAM	0xD0	mandatory
PROGRAM_RESET	0xCF	mandatory
GET_PGM_PROCESSOR_INFO	0xCE	optional
GET_SECTOR_INFO	0xCD	optional
PROGRAM_PREPARE	0xCC	optional
PROGRAM_FORMAT	0xCB	optional
PROGRAM_NEXT	0xCA	optional
PROGRAM_MAX	0xC9	optional
PROGRAM_VERIFY	0xC8	optional

8.5 Bit Mask Coded Parameters

Table 30 RESOURCE parameter in CONNECT and GET_SEED

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
×	×	×	PGM	STIM	DAQ	×	CAL/PAG

Table 31 COMM_MODE_BASIC parameter in CONNECT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OPTIONAL	SLAVE_BLOCK_MODE	x	x	x	ADDRESS_GRANULARITY_1	ADDRESS_GRANULARITY_0	BYTE_ORDER

Table 32 COMM_MODE_OPTIONAL parameter in GET_COMM_MODE_INFO

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	INTERLEAVED_MODE	MASTER_BLOCK_MODE

Table 33 COMM_MODE_PGM parameter in PROGRAM_START

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	SLAVE_BLOCK_MODE	x	x	x	x	INTERLEAVED_MODE	MASTER_BLOCK_MODE

Table 34 Mode parameter in GET_ID

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	COMPRESSED_ENCRYPTED	TRANSFER_MODE

Table 35 Current resource protection status parameter in GET_STATUS and UNLOCK

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	PGM	STIM	DAQ	x	CAL/PAG

Table 36 Mode parameter in SET_REQUEST

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	CLEAR_DAQ_REQ	STORE_DAQ_REQ_RESUME	STORE_DAQ_REQ_NO_RESUME	STORE_CAL_REQ

Table 37 Current session status parameter in GET_STATUS

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESUME	DAQ RUNNING	x	x	CLEAR_DAQ_REQ	STORE_DAQ_REQ	x	STORE_CAL_REQ

Table 38 DAQ_KEY_BYTE parameter in GET_DAQ_PROCESSOR_INFO

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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

Table 39 DAQ_PROPERTIES parameter in GET_DAQ_PROCESSOR_INFO

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OVERLOAD_EVENT	OVERLOAD_MSB	PID_OFF_SUPPORTED	TIMESTAMP_SUPPORTED	BIT_STIM_SUPPORTED	RESUME_SUPPORTED	PRESCALER_SUPPORTED	DAQ_CONFIG_TYPE

Table 40 Mode parameter in SET_DAQ_LIST_MODE

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	PID_OFF	TIMESTAMP	x	x	DIRECTION	ALTERNATING

Table 41 Current mode parameter in GET_DAQ_LIST_MODE

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESUME	RUNNING	PID_OFF	TIMESTAMP	x	x	DIRECTION	SELECTED

Table 42 DAQ_LIST_PROPERTIES parameter in GET_DAQ_LIST_INFO

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

Table 43 DAQ_EVENT_PROPERTIES parameter in GET_DAQ_EVENT_INFO

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

Table 44 TIMESTAMP_MODE parameter in GET_DAQ_RESOLUTION_INFO

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

Table 45 PAG_PROPERTIES parameter in GET_PAG_PROCESSOR_INFO

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	x	FREEZE_SUPPORTED

Table 46 Mode parameter in SET_SEGMENT_MODE

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	x	FREEZE

Table 47 Current mode parameter in GET_SEGMENT_MODE

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	x	FREEZE

Table 48 PAGE_PROPERTIES parameter in GET_PAGE_INFO

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	XCP_WRITE_ACCESS_WITH_ECU	XCP_WRITE_ACCESS_WITHOUT_ECU	XCP_READ_ACCESS_WITH_ECU	XCP_READ_ACCESS_WITHOUT_ECU	ECU_ACCESS_WITH_XCP	ECU_ACCESS_WITHOUT_XCP

Table 49 Mode parameter in SET_CAL_PAGE

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
All	x	x	x	x	x	XCP	ECU

Table 50 PGM_PROPERTIES parameter in GET_PGM_PROCESSOR_INFO

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NON_SEQ_PGM_REQUIRED	NON_SEQ_PGM_SUPPORTED	ENCRYPTION_REQUIRED	ENCRYPTION_SUPPORTED	COMPRESSION_REQUIRED	COMPRESSION_SUPPORTED	FUNCTIONAL_MODE	ABSOLUTE_MODE

8.6 Description of Commands

The following chapters are a description of all possible XCP 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:

Table 51 Command structure

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

Table 52 Command positive response structure

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

Table 53 Command negative response structure

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.

8.6.1 Standard Commands

8.6.1.1 Set up Connection With Slave

Category Standard, mandatory
Mnemonic CONNECT

Table 54 CONNECT command structure

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 XCP session (CONNECTED) this command has no influence on any configuration of the XCP 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 XCP communication with the slave.
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.

Positive Response:

Table 55 CONNECT positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	RESOURCE
2	BYTE	COMM_MODE_BASIC
3	BYTE	MAX_CTO, Maximum CTO size [BYTE]
4	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)

Table 56 RESOURCE parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	PGM	STIM	DAQ	x	CAL/PAG

Table 57 RESOURCE parameter bit mask coding

Flag	Description
CAL/PAG	CALibration and PAGing 0 = calibration/ paging not available 1 = calibration/ paging available
DAQ	DAQ lists supported 0 = DAQ lists not available 1 = DAQ lists 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

If a resource is available, the mandatory commands of this resource must be supported. For the allocation of commands to resources please refer to 1.4 Table of Command codes (CMD).

Regardless of the resource flag set, it may happen that the XCP handler cannot access the requested resource.
An Error Packet with `ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE` then will be sent to the master to indicate this situation.

Table 58 COMM_MODE_BASIC parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OPTIONAL	SLAVE_BLOCK_MODE	x	x	x	ADDRESS_GRANULARITY_1	ADDRESS_GRANULARITY_0	BYTE_ORDER

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

Table 59 COMM_MODE_BASIC parameter bit mask coding

Bit 2	Bit 1		
ADDRESS_GRANULARITY_1	**ADDRESS_GRANULARITY_0**	**ADDRESS_GRANULARITY**	**BYTE**
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.

Table 60 Data size dependency related to address granularity

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 mod AG = 0
   MAX_DTO mod 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 (XCP Protocol), it is based on bytes.

The XCP Protocol Layer Version Number indicates the major version of this Specification.

The XCP Transport Layer Version Number indicates the major version of the associated Transport Layer standard.

8.6.1.2 Disconnect From Slave

Category Standard, mandatory
Mnemonic DISCONNECT

Table 61 DISCONNECT command structure

Position	Type	Description
0	BYTE	Command Code = 0xFE

Brings the slave to the "DISCONNECTED" state.
The "DISCONNECTED" state is described in chapter State Machine.

Negative Response:

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

8.6.1.3 GET CURRENT SESSION STATUS FROM SLAVE

Category Standard, mandatory
Mnemonic GET_STATUS

Table 62 GET STATUS command structure

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:

Table 63 GET STATUS response structure

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

Table 64 Current session status parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESUME	DAQ_RUNNING	x	x	CLEAR_DAQ_REQ	STORE_DAQ_REQ	x	STORE_CAL_REQ

Table 65 Current session status parameter bit mask coding

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.

Table 66 Current resource protection status parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	PGM	STIM	DAQ	x	CAL/PAG

Table 67 Current resource protection status parameter bit mask coding

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
The Resource protection flags indicate that all commands allocated to the respective resource are protected and will return an ERR_ACCESS_LOCKED upon an attempt to execute the command without a previous successful GET_SEED/UNLOCK sequence.
For the allocation of commands to resources please refer to 1.4 Table of Command codes (CMD).

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.

8.6.1.4 Synchronize Command Execution After Time-out

Category Standard, mandatory

Mnemonic SYNCH

Table 68 SYNCH command structure

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:

Table 69 SYNCH negative response structure

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

8.6.1.5 GET COMMUNICATION MODE INFO

Category	Standard, optional
Mnemonic	GET_COMM_MODE_INFO

Table 70 GET COMM MODE INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xFB

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

Positive Response:

Table 71 GET COMM MODE INFO positive response structure

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

Table 72 COMM_MODE_OPTIONAL parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	INTERLEAVED_MODE	MASTER_BLOCK_MODE

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) 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 XCP Driver Version Number indicates the version number of the XCP driver in the slave.
The major driver version is the high nibble of the version number, the minor driver version is the low nibble.

8.6.1.6 GET IDENTIFICATION FROM SLAVE

Category Standard, optional

Mnemonic `GET_ID`

Table 73 GET ID command structure

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.

Table 74 Identification types

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:

Table 75 GET ID positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Mode
2	WORD	Reserved
4	DWORD	Length [BYTE]
8	BYTE 1	first byte of Identification (if mode = 1)
..	..	..
8+Length-1	BYTE n	nth byte of identification

The parameter Length specifies the number of bytes in the identification. If length is 0, the requested identification type is not available. The following rule applies: `Length mod AG = 0`

Table 76 GET_ID mode parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	COMPRESSED_ENCRYPTED	TRANSFER_MODE

If TRANSFER_MODE is 1, the identification is transferred in the remaining bytes of the response.
If TRANSFER_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. For the initial UPLOAD command, the following rule applies:

Number of Data Elements UPLOAD [AG] = (Length GET_ID [BYTE]) / AG

If COMPRESSED_ENCRYPTED is 1, the transferred data are compressed and/or encrypted. This is only allowed for Identification Type 4, i.e. “ASAM-MC2 file to upload”. The XCP master must decompress and/or decrypt the data using an implementation specific algorithm, implemented in an externally calculated function. The interface is described in chapter Interface to an External A2L Decompression/Decrypting Function.

The identification string is a byte stream of plain ASCII text, it does not have 0 termination. See table Table 226 GET_ID identification types for examples.

8.6.1.7 REQUEST TO SAVE TO NON-VOLATILE MEMORY

Category Standard, optional
Mnemonic SET_REQUEST

Table 77 SET REQUEST command structure

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

Table 78 SET_REQUEST mode parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	CLEAR_DAQ_REQ	STORE_DAQ_REQ_RESUME	STORE_DAQ_REQ_NO_RESUME	STORE_CAL_REQ

Table 79 SET_REQUEST mode parameter bit mask coding

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_NO_RESUME	REQuest to STORE DAQ list, no RESUME 0 = STORE_DAQ_REQ_NO_RESUME is not set. 1 = STORE_DAQ_REQ_NO_RESUME is set
STORE_DAQ_REQ_RESUME	REQuest to STORE DAQ list, RESUME enabled 0 = STORE_DAQ_REQ_RESUME is not set. 1 = STORE_DAQ_REQ_RESUME 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_x 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.

The STORE_DAQ_REQ_NO_RESUME does not set the slave into RESUME mode. The DAQ lists later on can be started by the XCP master at any time within an established XCP session.
The STORE_DAQ_REQ_RESUME sets a request to save all selected DAQ lists to memory, but at the same time implicitly sets the slave into RESUME mode.

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.

8.6.1.8 GET SEED FOR UNLOCKING A PROTECTED RESOURCE

Category Standard, optional (ref. UNLOCK)

Mnemonic GET_SEED

Table 80 GET SEED command structure

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: Do not 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:

Table 81 GET SEED positive response structure

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 is 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-2 MC description file. The API for communicating with the external function file is specified in chapter Interface to an External Seed&Key Function.

8.6.1.9 SEND KEY FOR UNLOCKING A PROTECTED RESOURCE

Category Standard, optional (ref. GET_SEED)
Mnemonic UNLOCK

Table 82 **UNLOCK command structure**

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:

Table 83 **UNLOCK positive response structure**

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

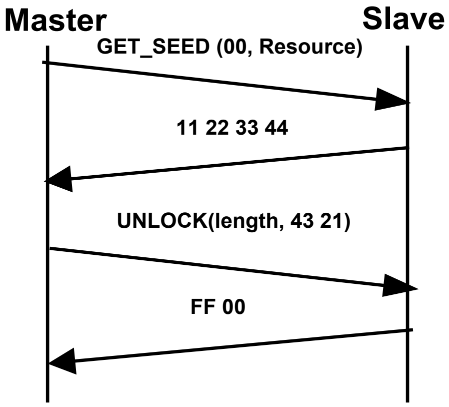

Figure 42 Short GET_SEED+UNLOCK sequence

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

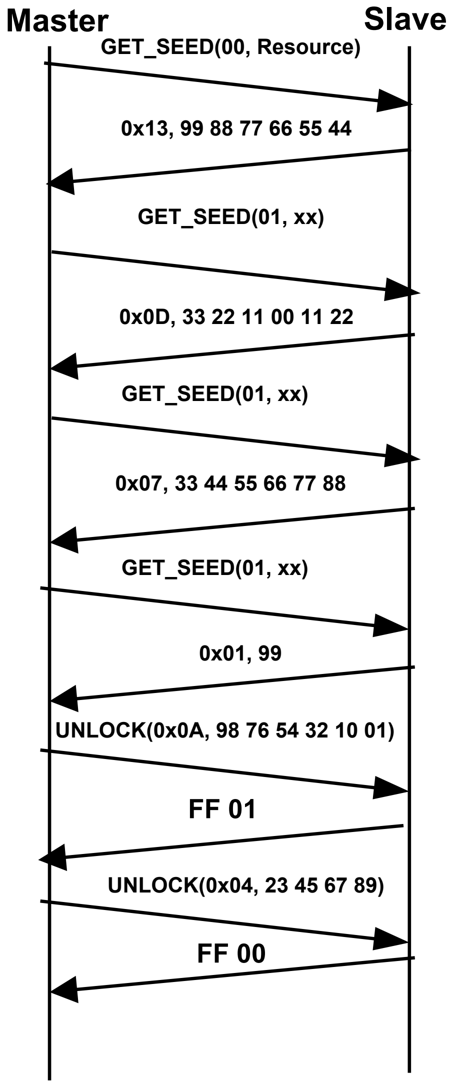

Figure 43 Long GET_SEED+UNLOCK sequence

8.6.1.10 Set Memory Transfer Address in Slave

Category Standard, optional

Mnemonic SET_MTA

Table 84 SET_MTA command structure

Position	Type	Description
0	BYTE	Command Code = 0xF6
1	WORD	Reserved
3	BYTE	Address extension
4	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_NEXT, DOWNLOAD_MAX, MODIFY_BITS, PROGRAM_CLEAR, PROGRAM, PROGRAM_NEXT and PROGRAM_MAX.

8.6.1.11 Upload From Slave to Master

Category Standard, optional
Mnemonic UPLOAD

Table 85 **UPLOAD command structure**

Position	Type	Description
0	BYTE	Command Code = 0xF5
1	BYTE	n = 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:

Table 86 **UPLOAD positive response structure**

Position	Type	Description
0	BYTE	Packet ID: 0xFF
..	BYTEs	Used for alignment only if AG > 1
AG	ELEMENT 1	1st data element
..	..	..
n*AG	ELEMENT n	nth data element

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/AG-1]. An ERR_OUT_OF_RANGE will be returned, if the number of data elements is more than MAX_CTO/AG-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)-1) / (MAX_CTO-AG) +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
```

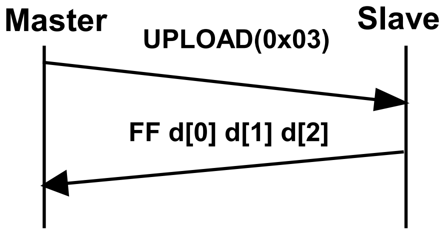

Figure 44 UPLOAD 3 bytes

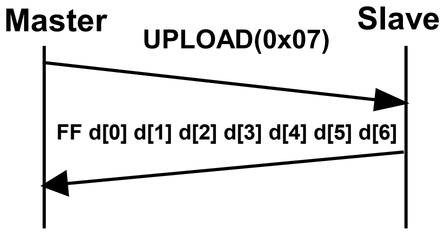

Figure 45 UPLOAD 7 bytes

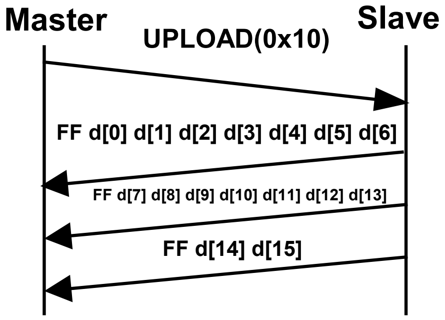
Master
Slave
UPLOAD(0x10)
FF d[0] d[1] d[2] d[3] d[4] d[5] d[6]
FF d[7] d[8] d[9] d[10] d[11] d[12] d[13]
FF d[14] d[15]

Figure 46 UPLOAD 16 bytes in block mode

8.6.1.12 UPLOAD FROM SLAVE TO MASTER (SHORT VERSION)

Category Standard, optional

Mnemonic SHORT_UPLOAD

Table 87 SHORT UPLOAD command structure

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

8.6.1.13 Build Checksum Over Memory Range

Category Standard, optional

Mnemonic BUILD_CHECKSUM

Table 88 BUILD CHECKSUM command structure

Position	Type	Description
0	BYTE	Command Code = 0xF3
1	BYTE	reserved
2	WORD	reserved
4	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:

Table 89 BUILD CHECKSUM positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	Checksum type
2	WORD	Reserved
4	DWORD	Checksum

Table 90 Checksum types

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
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
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-2 MC description file.
The API for communicating with the external function file is specified in chapter Interface to an External Checksum Function.

Negative Response:

Table 91 BUILD CHECKSUM negative response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFE
1	BYTE	Error code
2	WORD	reserved
4	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.

Table 92 CRC algorithm parameter overview

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: [5]

The following tables provide information for validating the checksum calculation algorithms.
The test pattern is the hexadecimal representation of the contents of a 32-byte binary file/data stream, starting with the lowest address, ending with the highest address.

Table 93 Test pattern

Test pattern
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x10 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE 0xFF 0x00

Table 94 Checksum results for different checksum types

Name	Expected checksum Intel	Expected checksum Motorola
XCP_ADD_11	0x10	0x10
XCP_ADD_12	0x0F10	0x0F10
XCP_ADD_14	0x00000F10	0x00000F10
XCP_ADD_22	0x1800	0x0710
XCP_ADD_24	0x00071800	0x00080710
XCP_ADD_44	0x140C03F8	0xFC040B10
XCP_CRC_16	0xC76A	0xC76A
XCP_CRC_16_CITT	0x9D50	0x9D50
XCP_CRC_32	0x89CD97CE	0x89CD97CE

8.6.1.14 Refer to Transport Layer Specific Command

Category Standard, auxiliary

Mnemonic `TRANSPORT_LAYER_CMD`

Table 95 TRANSPORT LAYER CMD structure

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

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

Example:

Category CAN only, optional

Mnemonic `GET_SLAVE_ID`

8.6.1.15 Refer to User-defined Command

Category Standard, auxiliary

Mnemonic `USER_CMD`

Table 96 USER CMD structure

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

This command is user-defined. It must not be used to implement functionalities done by other services.

8.6.2 Calibration Commands

8.6.2.1 Download From Master to Slave

Category Calibration, mandatory

Mnemonic DOWNLOAD

Table 97 DOWNLOAD command structure

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

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 elements.
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/AG-2]. An ERR_OUT_OF_RANGE will be returned, if the number of data elements is more than MAX_CTO/AG-2.
After receiving a DOWNLOAD command the XCP slave first has to check whether there are enough resources available in order to cover the complete download request. If the XCP slave does not have enough resources, it has to send ERR_MEMORY_OVERFLOW and does not execute any single download request. If a DOWNLOAD request will be rejected, there have been no changes to the slave's memory contents at all.
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)].
If AG=1 the master device has to transmit ((n*AG)-1) / (MAX_CTO-2)) additional consecutive DOWNLOAD_NEXT command packets.
If AG>1 the master device has to transmit ((n*AG)-1) / (MAX_CTO-AG)) additional consecutive DOWNLOAD_NEXT command packets.
Without any error, 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 `GET_COMM_MODE_INFO` command (`MAX_BS`, `MIN_ST`).
If the XCP slave detects an internal problem during a block mode transfer, it can send a negative response at once. If block transfer mode is requested and not enough resources are available, the XCP slave can send the negative response code already after the initial `DOWNLOAD` command of the XCP master.

Example:

```
MAX_CTO=8
```

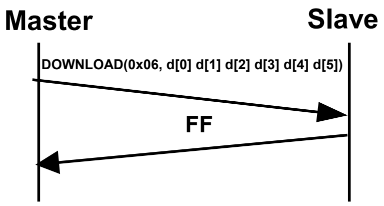

Figure 47 DOWNLOAD 6 bytes

8.6.2.2 DOWNLOAD FROM MASTER TO SLAVE (BLOCK MODE)

Category Calibration, optional
Mnemonic DOWNLOAD_NEXT

Table 98 DOWNLOAD NEXT command structure

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

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.

Table 99 DOWNLOAD NEXT negative response structure

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

Example:

MAX_CTO=8

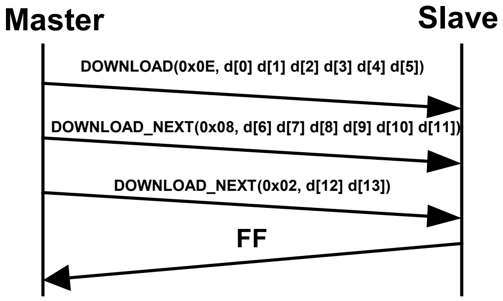
Master
Slave
DOWNLOAD(0x0E, d[0] d[1] d[2] d[3] d[4] d[5])
DOWNLOAD_NEXT(0x08, d[6] d[7] d[8] d[9] d[10] d[11])
DOWNLOAD_NEXT(0x02, d[12] d[13])
FF

Figure 48 DOWNLOAD 14 bytes in block mode

8.6.2.3 DOWNLOAD FROM MASTER TO SLAVE (FIXED SIZE)

Category Calibration, optional

Mnemonic DOWNLOAD_MAX

Table 100 DOWNLOAD MAX command structure

Position	Type	Description
0	BYTE	Command Code = 0xEE
..	BYTEs	Used for alignment, only if AG >1
AG	ELEMENT 1	1st data element
..	..	..
MAX_CTO-AG	ELEMENT n	nth data element

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 n 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.

After receiving a DOWNLOAD_MAX command the XCP slave first has to check whether there are enough resources available in order to cover the complete download request. If the XCP slave does not have enough resources, it has to send ERR_MEMORY_OVERFLOW and does not execute any single download request. If a DOWNLOAD_MAX request will be rejected, there have been no changes to the slave’s memory contents at all.

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

8.6.2.4 DOWNLOAD FROM MASTER TO SLAVE (SHORT VERSION)

Category Calibration, optional

Mnemonic SHORT_DOWNLOAD

Table 101 SHORT DOWNLOAD command structure

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	DWORD	Address
8	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.
After receiving a SHORT_DOWNLOAD command the XCP slave first has to check whether there are enough resources available in order to cover the complete download request. If the XCP slave does not have enough resources, it has to send ERR_MEMORY_OVERFLOW and does not execute any single download request. If a SHORT_DOWNLOAD request will be rejected, there have been no changes to the slave's memory contents at all.

This command does not support block transfer and it must not 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. XCP on CAN).

8.6.2.5 MODIFY BITS

Category   Calibration, optional
Mnemonic   MODIFY_BITS

Table 102 MODIFY BITS command structure

Position	Type	Description
0	BYTE	Command Code = 0xEC
1	BYTE	Shift Value (S)
2	WORD	AND Mask (MA)
4	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	0	1111	1111	1111	1111

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

S = 16
A = 1111 1111 1111 0000 1111 1111 1111 1111
MA = 1011 1111 1111 1110
MX = 0000 0000 0000 0001

Result:
A = 1011 1111 1111 0001 1111 1111 1111 1111

The MTA will not be affected.

8.6.3 PAGE SWITCHING COMMANDS

8.6.3.1 SET CALIBRATION PAGE

Category	Page switching, optional
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).

Table 103 SET CAL PAGE command structure

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.

Table 104 SET CAL PAGE mode parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
All	x	x	x	x	x	XCP	ECU

Table 105 Mode parameter bit explanation

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

Both flags ECU and XCP 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.

8.6.3.2 Get Calibration Page

Category Page switching, optional

Mnemonic GET_CAL_PAGE

Table 106 GET CAL PAGE command structure

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 (XCP access). All other values are invalid.

Positive Response:

Table 107 GET CAL PAGE positive response structure

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

8.6.3.3 Get General Information on PAG Processor

Category Paging, optional

Mnemonic GET_PAG_PROCESSOR_INFO

Table 108 GET PAG PROCESSOR_INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xE9

This command returns general information on paging.

Positive response:

Table 109 GET PAG PROCESSOR_INFO positive response structure

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

Table 110 PAG_PROPERTIES parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
×	×	×	×	×	×	×	FREEZE_SUPPORTED

Table 111 PAG_PROPERTIES parameter bit mask coding

Flag	Description
FREEZE_SUPPORTED	0 = SEGMENTS cannot 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.

8.6.3.4 GET SPECIFIC INFORMATION FOR A SEGMENT

Category Page switching, optional
Mnemonic GET_SEGMENT_INFO

Table 112 GET SEGMENT INFO command structure

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: do not care Mode 2: 0 = source address 1 = destination address 2 = length address
4	BYTE	MAPPING_INDEX [0,1,..MAX_MAPPING-1] Mode 0: do not care Mode 1: do not 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 “do not care”.
For Mode = 2, MAPPING_INDEX indicates the range MAPPING_INFO belongs to.
For Mode = 0 and Mode = 1, MAPPING_INDEX is “do not 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)

Table 113 GET SEGMENT INFO positive response structure (mode 0)

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	reserved
2	WORD	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)

Table 114 GET SEGMENT INFO positive response structure (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)

Table 115 GET SEGMENT INFO positive response structure (mode 2)

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	reserved
2	WORD	reserved

Position	Type	Description
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.

8.6.3.5 GET SPECIFIC INFORMATION FOR A PAGE

Category Page switching, optional

Mnemonic GET_PAGE_INFO

Table 116 GET PAGE INFO command structure

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:

Table 117 GET PAGE INFO positive response structure

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

Table 118 Page properties parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	XCP_WRITE_ACCESS_WITH_ECU	XCP_WRITE_ACCESS_WITHOUT_ECU	XCP_READ_ACCESS_WITH_ECU	XCP_READ_ACCESS_WITHOUT_ECU	ECU_ACCESS_WITH_XCP	ECU_ACCESS_WITHOUT_XCP

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 does not care whether the XCP master accesses this page at the same time or not.

Table 119 ECU access type coding

Bit 1	Bit 0	
ECU_ACCESS_WITH_XCP	**ECU_ACCESS_WITHOUT_XCP**	**ECU_ACCESS_TYPE**
0	0	ECU access not allowed
0	1	without XCP only
1	0	with XCP only
1	1	do not care

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).

Table 120 XCP master read access type coding

Bit 3	Bit 2	
XCP_READ_ACCESS_WITH_ECU	XCP_READ_ACCESS_WITHOUT_ECU	XCP_READ_ACCESS_TYPE
0	0	XCP READ access not allowed
0	1	without ECU only
1	0	with ECU only
1	1	do not care

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 does not need to care whether the ECU accesses this page at the same time or not.

Table 121 XCP master write access type coding

Bit 5	**Bit 4**	
XCP_WRITE_ACCESS_WITH_ECU	**XCP_WRITE_ACCESS_WITHOUT_ECU**	**XCP_WRITE_ACCESS_TYPE**
0	0	XCP WRITE access not allowed
0	1	without ECU only
1	0	with ECU only
1	1	do not care

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 does not 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.

8.6.3.6 SET MODE FOR A SEGMENT

Category Page switching, optional
Mnemonic SET_SEGMENT_MODE

Table 122 SET SEGMENT MODE command structure

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.

Table 123 SET SEGMENT MODE parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	x	FREEZE

Table 124 Freeze bit description

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

The FREEZE flag selects the SEGMENT for freezing through STORE_CAL_REQ.

8.6.3.7 GET MODE FOR A SEGMENT

Category Page switching, optional

Mnemonic GET_SEGMENT_MODE

Table 125 GET SEGMENT MODE command structure

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:

Table 126 GET SEGMENT MODE positive response structure

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

Table 127 GET SEGMENT MODE parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	x	FREEZE

8.6.3.8 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.

Table 128 COPY CAL PAGE command structure

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.

8.6.4 DATA ACQUISITION AND STIMULATION COMMANDS

8.6.4.1 SET POINTER TO ODT ENTRY

Category	Data acquisition and stimulation, basic, mandatory
Mnemonic	SET_DAQ_PTR

Table 129 SET DAQ PTR command structure

Position	Type	Description
0	BYTE	Command Code = 0xE2
1	BYTE	Reserved
2	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.

8.6.4.2 WRITE ELEMENT IN ODT ENTRY

Category Data acquisition and stimulation, basic, mandatory

Mnemonic WRITE_DAQ

Table 130 WRITE DAQ command structure

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	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.

8.6.4.3 Set Mode for DAQ List

Category Data acquisition and stimulation, basic, mandatory
Mnemonic SET_DAQ_LIST_MODE

Table 131 SET DAQ LIST MODE command structure

Position	Type	Description
0	BYTE	Command Code = 0xE0
1	BYTE	Mode
2	WORD	DAQ_LIST_NUMBER [0,1,..MAX_DAQ-1]
4	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.

Table 132 SET DAQ LIST MODE parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	PID_OFF	TIMESTAMP	x	x	DIRECTION	ALTERNATING

Table 133 SET DAQ LIST MODE parameter bit mask coding

Flag	Description
ALTERNATING	0 = disable alternating display mode 1 = enable alternating display mode
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 configures the DAQ list for synchronized data acquisition or synchronized data stimulation mode.

The ALTERNATING flag selects the alternating display mode. When this flag is set, the master must assign this DAQ list to the special event channel specified by

DAQ_ALTERNATING_SUPPORTED in the ASAM MCD 2MC description file. The slave may support up to one event channel for 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-2 MC description file is not allowed to use TIMESTAMP_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.

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.

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 XCP 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 does not support the requested prioritization of DAQ lists, this will be indicated by returning ERR_OUT_OF_RANGE.

8.6.4.4 START/STOP/SELECT DAQ LIST

Category Data acquisition and stimulation, basic, mandatory
Mnemonic START_STOP_DAQ_LIST

Table 134 START STOP DAQ LIST command structure

Position	Type	Description
0	BYTE	Command Code = 0xDE
1	BYTE	Mode
		00 = stop
		01 = start
		02 = select
2	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 storing DAQ lists (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:

Table 135 START STOP DAQ LIST positive response structure

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.

8.6.4.5 START/STOP DAQ LISTS (SYNCHRONOUSLY)

Category Data acquisition and stimulation, basic, mandatory
Mnemonic START_STOP_SYNCH

Table 136 START STOP SYNCH command structure

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.

8.6.4.6 WRITE MULTIPLE ELEMENTS IN ODT

Category Data acquisition and stimulation, basic, optional
Mnemonic WRITE_DAQ_MULTIPLE

Table 137 WRITE DAQ MULTIPLE command structure

Position	Type	Description
0	BYTE	Command Code = 0xC7
1	BYTE	n = NoDAQ, number of consecutive DAQ elements
2	BYTE	BIT_OFFSET [0..31] of 1st element Position of bit in 32-bit variable referenced by the address and extension below
3	BYTE	Size of 1st DAQ element 0 <= size <= MAX_ODT_ENTRY_SIZE_DAQ_x
4	DWORD	Address of 1st DAQ element
8	BYTE	Address extension of 1st DAQ element
9	BYTE	Dummy for alignment of the next element
10	BYTE	BIT_OFFSET [0..31] of 2nd element Position of bit in 32-bit variable referenced by the address and extension below
11	BYTE	Size of 2nd DAQ element 0 <= size <= MAX_ODT_ENTRY_SIZE_DAQ_x
12	DWORD	Address of 2nd DAQ element
16	BYTE	Address extension of 2nd DAQ element
17	BYTE	Dummy for alignment of the next element
..	..	..
(n-1)*8+2	BYTE	BIT_OFFSET [0..31] of nth element Position of bit in 32-bit variable referenced by the address and extension below
(n-1)*8+3	BYTE	Size of nth DAQ element 0 <= size <= MAX_ODT_ENTRY_SIZE_DAQ_x
(n-1)*8+4	DWORD	Address of nth DAQ element
n*8	BYTE	Address extension of nth DAQ element
n*8+1	BYTE	Dummy of the last element

This command is used to write consecutive ODT entries to a DAQ list defined by the DAQ list pointer (see SET_DAQ_PTR).

NoDAQ is the number of consecutively written DAQ elements. NoDAQ is limited by the maximum command packet size MAX_CTO.

The dummy byte at the end of each DAQ element must be used for alignment issues, even for the last element.

In general WRITE_DAQ_MULTIPLE has the same restrictions as the WRITE_DAQ command.
All DAQ entries within one WRITE_DAQ_MULTIPLE must be written into one ODT.
WRITE_DAQ_MULTIPLE must not be used to write over ODT borders.

The error handling is identical to the one for WRITE_DAQ. However, it is not possible to detect which entry caused the error. In that case the whole configuration is invalid.

If the optional command WRITE_DAQ_MULTIPLE is used, the requirement MAX_CTO>=10 has to be fulfilled.

8.6.4.7 READ ELEMENT FROM ODT ENTRY

Category Data acquisition and stimulation, basic, optional

Mnemonic READ_DAQ

Table 138 READ DAQ command structure

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:

Table 139 READ DAQ positive response structure

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	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).

8.6.4.8 GET DAQ CLOCK FROM SLAVE

Category Data acquisition and stimulation, basic, optional

Mnemonic GET_DAQ_CLOCK

Table 140 GET DAQ CLOCK command structure

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:

Table 141 GET DAQ CLOCK positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	reserved
2	WORD	reserved
4	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

8.6.4.9 GET GENERAL INFORMATION ON DAQ PROCESSOR

Category Data acquisition and stimulation, basic, optional

Mnemonic GET_DAQ_PROCESSOR_INFO

Table 142 GET DAQ PROCESSOR INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xDA

This command returns general information on DAQ lists.

Positive Response:

Table 143 GET DAQ PROCESSOR INFO positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	DAQ_PROPERTIES General properties of DAQ lists
2	WORD	MAX_DAQ Total number of available DAQ lists
4	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

Table 144 DAQ properties parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OVERLOAD_EVENT	OVERLOAD_MSB	PID_OFF_SUPPORTED	TIMESTAMP_SUPPORTED	BIT_STIM_SUPPORTED	RESUME_SUPPORTED	PRESCALER_SUPPORTED	DAQ_CONFIG_TYPE

Table 145 DAQ properties parameter bit mask coding

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 does not 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 overload indication method:

Table 146 Overload bit mask coding

Bit 7	Bit 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 overload 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 overload 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.

Table 147 DAQ key byte parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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

Table 148 Optimisation bit mask coding

Bit 3	Bit 2	Bit 1	Bit 0	
Optimisation_Type_3	Optimisation_Type_2	Optimisation_Type_1	Optimisation_Type_0	Optimisation Type
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.

Table 149 Address extension bit mask coding

Bit 5	Bit 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.

Table 150 Identification field type bit mask coding

Bit 7	Bit 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”.

8.6.4.10 Get General Information on DAQ Processing Resolution

Category Data acquisition and stimulation, basic, optional

Mnemonic `GET_DAQ_RESOLUTION_INFO`

Table 151 GET DAQ RESOLUTION INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xD9

This command returns information on the resolution of DAQ lists.

Positive Response:

Table 152 GET DAQ RESOLUTION INFO positive response structure

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	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 does not 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 and wraps around if an overflow occurs.

$$t^{k+1}_{physical} = t^{k}_{physical} + [(t^{k+1}_{protocol} - t^{k}_{protocol}) * TIMESTAMP\_UNIT * TIMESTAMP\_TICKS]$$

Table 153 Timestamp mode parameter bit mask structure

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

Table 154 Size parameter bit mask coding

Bit 2	**Bit 1**	**Bit 0**	
Size_2	**Size_1**	**Size_0**	**Timestamp size [bytes]**
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.

Table 155 Unit parameter bit mask coding

Bit 7	Bit 6	Bit 5	Bit 4	
Unit_3	Unit_2	Unit_1	Unit_0	Timestamp unit
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
1	0	1	0	DAQ_TIMESTAMP_UNIT_1PS 1 PS = 1 picosecond = 10^{-12} second
1	0	1	1	DAQ_TIMESTAMP_UNIT_10PS
1	1	0	0	DAQ_TIMESTAMP_UNIT_100PS

8.6.4.11 Get Mode From DAQ List

Category Data acquisition and stimulation, basic, optional

Mnemonic `GET_DAQ_LIST_MODE`

Table 156 GET DAQ LIST MODE command structure

Position	Type	Description
0	BYTE	Command Code = 0xDF
1	BYTE	Reserved
2	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:

Table 157 GET DAQ LIST MODE positive response structure

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

Table 158 Current Mode parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESUME	RUNNING	PID_OFF	TIMESTAMP	x	x	DIRECTION	SELECTED

Table 159 Current Mode parameter bit mask coding

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.

8.6.4.12 Get Specific Information for an Event Channel

Category Data acquisition and stimulation, basic, optional

Mnemonic GET_DAQ_EVENT_INFO

Table 160 GET DAQ EVENT INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xD7
1	BYTE	Reserved
2	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:

Table 161 GET DAQ EVENT INFO positive response structure

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 do not care if Event channel time cycle = 0
6	BYTE	EVENT_CHANNEL_PRIORITY (FF highest)

Table 162 DAQ_EVENT_PROPERTIES parameter bit mask structure

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

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

Table 163 GET DAQ EVENT INFO DAQ/STIM parameter bit mask coding

Bit 3	Bit 2	
STIM	**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)

The CONSISTENCY_DAQ flag indicates that for this Event Channel all data that belong to one and the same DAQ list are processed consistently.

The CONSISTENCY_EVENT flag indicates that for this Event Channel all data are processed consistently.

Table 164 Consistency parameter bit mask coding

Bit 7	Bit 6	
CONSISTENCY_EVENT	CONSISTENCY_DAQ	CONSISTENCY
0	0	Consistency on ODT level (default)
0	1	Consistency on DAQ list level
1	0	Consistency on Event Channel level

If there is only one DAQ list associated with this Event Channel, CONSISTENCY_DAQ has the same meaning as CONSISTENCY_EVENT.

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

If more than one DAQ list is associated with this Event Channel, CONSISTENCY_EVENT implies that all data of DAQ lists in this Event Channel are processed consistently.

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 cannot be used. MAX_DAQ_LIST = 0xFF means there is 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. For the initial UPLOAD command, the following rule applies:

Number of Data Elements UPLOAD [AG] = (Length GET_DAQ_EVENT_INFO [BYTE]) / AG

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.
The EVENT_CHANNEL_TIME_UNIT is defined as follows:

Table 165 Event channel time unit coding

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
x	**x**	**x**	**x**	**Unit_3**	**Unit_2**	**Unit_1**	**Unit_0**	**EVENT_CHANNEL_TIME_UNIT**
x	x	x	x	0	0	0	0	EVENT_CHANNEL _TIME_UNIT_1NS 1 NS = 1 nanosecond = 10^{-9} second
x	x	x	x	0	0	0	1	EVENT_CHANNEL _TIME_UNIT_10NS
x	x	x	x	0	0	1	0	EVENT_CHANNEL _TIME_UNIT_100NS
x	x	x	x	0	0	1	1	EVENT_CHANNEL _TIME_UNIT_1US 1 US = 1 microsecond = 10^{-6} second
x	x	x	x	0	1	0	0	EVENT_CHANNEL _TIME_UNIT_10US
x	x	x	x	0	1	0	1	EVENT_CHANNEL _TIME_UNIT_100US
x	x	x	x	0	1	1	0	EVENT_CHANNEL _TIME_UNIT_1MS 1 MS = 1 millisecond = 10^{-3} second
x	x	x	x	0	1	1	1	EVENT_CHANNEL _TIME_UNIT_10MS
x	x	x	x	1	0	0	0	EVENT_CHANNEL _TIME_UNIT_100MS
x	x	x	x	1	0	0	1	EVENT_CHANNEL _TIME_UNIT_1S 1 S = 1 second = 1 second
x	x	x	x	1	0	1	0	EVENT_CHANNEL _TIME_UNIT_1PS 1 PS = 1 picosecond = 10^{-12} second
x	x	x	x	1	0	1	1	EVENT_CHANNEL _TIME_UNIT_10PS
x	x	x	x	1	1	0	0	EVENT_CHANNEL _TIME_UNIT_100PS

Please note that the EVENT_CHANNEL_TIME_UNIT is coded in the lower nibble of the parameter. This coding differs from the one used for the Timestamp unit, which is in the higher nibble of the parameter.
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

8.6.4.13 CLEAR DAQ LIST CONFIGURATION

Category Data acquisition and stimulation, static, mandatory

Mnemonic CLEAR_DAQ_LIST

Table 166 CLEAR DAQ LIST command structure

Position	Type	Description
0	BYTE	Command Code = 0xE3
1	BYTE	reserved
2	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.

8.6.4.14 Get Specific Information for a DAQ List

Category	Data acquisition and stimulation, static, optional
Mnemonic	GET_DAQ_LIST_INFO

Table 167 GET DAQ LIST INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xD8
1	BYTE	reserved
2	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:

Table 168 GET DAQ LIST INFO positive response structure

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	WORD	FIXED_EVENT Number of the fixed event channel for this DAQ list

Table 169 DAQ_LIST_PROPERTIES parameter bit mask structure

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

Table 170 BIT 0/Bit 1 parameter bit mask coding

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 cannot 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 cannot be changed.
The `DAQ` and `STIM` flags indicate which DIRECTION can be used for this DAQ list

Table 171 GET DAQ LIST INFO DAQ/STIM parameter bit mask coding

Bit 3	Bit 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.

8.6.4.15 CLEAR DYNAMIC DAQ CONFIGURATION

Category	Data acquisition and stimulation, dynamic, optional
Mnemonic	FREE_DAQ

Table 172 FREE DAQ command structure

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.

8.6.4.16 ALLOCATE DAQ LISTS

Category Data acquisition and stimulation, dynamic, optional

Mnemonic ALLOC_DAQ

Table 173 ALLOC_DAQ command structure

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

This command allocates a number of DAQ lists for this XCP slave device.
If there is 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.

8.6.4.17 Allocate ODTs to a DAQ List

Category	Data acquisition and stimulation, dynamic, optional
Mnemonic	ALLOC_ODT

Table 174 ALLOC ODT command structure

Position	Type	Description
0	BYTE	Command Code = 0xD4
1	BYTE	Reserved
2	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 is 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.

8.6.4.18 Allocate ODT Entries to an ODT

Category Data acquisition and stimulation, dynamic, optional

Mnemonic ALLOC_ODT_ENTRY

Table 175 ALLOC_ODT_ENTRY command structure

Position	Type	Description
0	BYTE	Command Code = 0xD3
1	BYTE	Reserved
2	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 is 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.

8.6.5 Non-Volatile Memory Programming

8.6.5.1 Indicate the Beginning of a Programming Sequence

Category Non-volatile memory programming, mandatory

Mnemonic PROGRAM_START

Table 176 PROGRAM_START command structure

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, etc.) and the execution of other commands may be restricted during a programming sequence (data acquisition may not run, calibration may be restricted, etc.). 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:

Table 177 PROGRAM_START positive response structure

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

Table 178 COMM_MODE_PGM parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	SLAVE_BLOCK_MODE	x	x	x	x	INTERLEAVED_MODE	MASTER_BLOCK_MODE

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.

8.6.5.2 CLEAR A PART OF NON-VOLATILE MEMORY

Category Non-volatile memory programming, mandatory
Mnemonic PROGRAM_CLEAR

Table 179 PROGRAM_CLEAR command structure

Position	Type	Description
0	BYTE	Command Code = 0xD1
1	BYTE	Mode
2	WORD	reserved
4	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.

Table 180 Mode parameter byte structure

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

Table 181 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-2 MC 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.

Table 182 **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.

8.6.5.3 PROGRAM A NON-VOLATILE MEMORY SEGMENT

Category Non-volatile memory programming, mandatory

Mnemonic PROGRAM

Table 183 PROGRAM command structure

Position	Type	Description
0	BYTE	Command Code = 0xD0
1	BYTE	n = Number of data elements [AG] [1..(MAX_CTO_PGM-2)/AG]
..	BYTEs	Used for alignment, only if AG = 4
AG=1: 2 AG>1: AG	ELEMENT 1	1st data element
..	..	..
AG=1: n+1 AG>1: n*AG	ELEMENT n	nth data element

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.
If the slave device does not support block transfer mode, all programmed 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_PGM/AG-2]. An ERR_OUT_OF_RANGE will be returned, if the number of data elements is more than MAX_CTO_PGM/AG-2.
If block transfer mode is supported, the programmed 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_PGM).
Therefore the number of data elements (n) can be in the range [1..min(MAX_BS_PGM*(MAX_CTO_PGM-2)/AG,255)].
If AG=1 the master device has to transmit ((n*AG)-1) / (MAX_CTO_PGM-2)) additional, consecutive PROGRAM_NEXT command packets.
If AG>1 the master device has to transmit ((n*AG)-1) / (MAX_CTO_PGM-AG)) additional, consecutive PROGRAM_NEXT command packets.
The slave device will acknowledge only the last PROGRAM_NEXT command packet. The separation time between the command packets and the maximum number of packets are specified in the response for the PROGRAM_START command (MAX_BS_PGM, MIN_ST_PGM).

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.

8.6.5.4 INDICATE THE END OF A PROGRAMMING SEQUENCE

Category Non-volatile memory programming, mandatory

Mnemonic PROGRAM_RESET

Table 184 PROGRAM_RESET command structure

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.

8.6.5.5 Get General Information on PGM Processor

Category Programming, optional

Mnemonic GET_PGM_PROCESSOR_INFO

Table 185 GET PGM PROCESSOR INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xCE

This command returns general information on programming.

Positive response:

Table 186 GET PGM PROCESSOR INFO positive response structure

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

Table 187 PGM_PROPERTIES parameter bit mask structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NON_SEQ_PGM_REQUIRED	NON_SEQ_PGM_SUPPORTED	ENCRYPTION_REQUIRED	ENCRYPTION_SUPPORTED	COMPRESSION_REQUIRED	COMPRESSION_SUPPORTED	FUNCTIONAL_MODE	ABSOLUTE_MODE

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

Table 188 PGM_PROPERTIES mode parameter bit mask coding

Bit 1	Bit 0	
FUNCTIONAL_MODE	ABSOLUTE_MODE	**clear/programming 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.

Table 189 Compression parameter bit mask coding

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

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.

Table 190 Encryption parameter bit mask coding

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

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.

Table 191 Non sequential programming parameter bit mask coding

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

MAX_SECTOR is the total number of sectors in the slave device

8.6.5.6 GET SPECIFIC INFORMATION FOR A SECTOR

Category Programming, optional

Mnemonic GET_SECTOR_INFO

Table 192 GET SECTOR INFO command structure

Position	Type	Description
0	BYTE	Command Code = 0xCD
1	BYTE	Mode 0 = get start address for this SECTOR 1 = get length of this SECTOR [BYTE] 2 = get name 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 (mode = 0 or mode = 1):

Table 193 GET SECTOR INFO positive response structure (mode = 0 or 1)

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

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 in bytes the length of this SECTOR in `SECTOR_INFO`.
The following rule applies: `Length mod AG = 0`.

Positive response (mode = 2):

Table 194 GET SECTOR INFO positive response structure (mode = 2)

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	`SECTOR_NAME_LENGTH` in bytes 0 – if not available

If `Mode = 2`, this command automatically sets the Memory Transfer Address (MTA) to the location from which the Master may upload the SECTOR name as ASCII text, using one or more `UPLOAD` commands. For the initial `UPLOAD` command, the following rule applies:

```
Number of Data Elements UPLOAD [AG] = (Length GET_SECTOR_INFO
[BYTE]) / AG
```

The `SECTOR_NAME_LENGTH` specifies the number of ASCII bytes in the name. There must be no 0 termination.

8.6.5.7 PREPARE NON-VOLATILE MEMORY PROGRAMMING

Category Non-volatile memory programming, optional

Mnemonic PROGRAM_PREPARE

Table 195 PROGRAM PREPARE command structure

Position	Type	Description
0	BYTE	Command Code = 0xCC
1	BYTE	Not used
2	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.

8.6.5.8 SET DATA FORMAT BEFORE PROGRAMMING

Category Non-volatile memory programming, optional

Mnemonic `PROGRAM_FORMAT`

Table 196 PROGRAM FORMAT command structure

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 directly at the beginning of the programming sequence and is valid until the end of this sequence. The sequence will be terminated by other commands e.g. `SET_MTA`.

If this command is not 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`.

Table 197 Reformatting method

Parameter	Hex	Description
Compression method	0x00	Data uncompressed (default)
	0x80...0xFF	User-defined
Encryption method	0x00	Data not encrypted (default)
	0x80...0xFF	User-defined
Programming method	0x00	Sequential Programming (default)
	0x80...0xFF	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-2 MC 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
```

8.6.5.9 PROGRAM A NON-VOLATILE MEMORY SEGMENT (BLOCK MODE)

Category Non-volatile memory programming, optional

Mnemonic PROGRAM_NEXT

Table 198 PROGRAM NEXT command structure

Position	Type	Description
0	BYTE	Command Code = 0xCA
1	BYTE	n = Number of data elements [AG] [1..(MAX_CTO_PGM-2)/AG]
..	BYTEs	Used for alignment, only if AG = 4
AG=1: 2 AG>1: AG	ELEMENT 1	1st data element
..	..	..
AG=1: n+1 AG>1:n* AG	ELEMENT n	nth data element

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.

Table 199 PROGRAM NEXT negative response structure

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

8.6.5.10 Program a Non-volatile Memory Segment (fixed size)

Category Non-volatile memory programming, optional

Mnemonic PROGRAM_MAX

Table 200 **PROGRAM MAX command structure**

Position	Type	Description
0	BYTE	Command Code = 0xC9
..	BYTEs	Used for alignment, only if AG >1
AG	ELEMENT 1	1st data element
..	..	..
MAX_CTO_PGM-AG	ELEMENT n	nth data element

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_PGM-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_PGM-1.
This command does not support block transfer and it must not be used within a block transfer sequence.

8.6.5.11 PROGRAM VERIFY

Category Non-volatile memory programming, optional

Mnemonic PROGRAM_VERIFY

Table 201 PROGRAM VERIFY command structure

Position	Type	Description
0	BYTE	Command Code = 0xC8
1	BYTE	Verification mode 00 = request to start internal routine 01 = sending Verification Value
2	WORD	Verification Type
4	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 XCP parameters allow a wide range of project specific adaptions.

Table 202 Verification type parameter bit mask structure

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.

8.7 Communication Error Handling

8.7.1 Definitions

8.7.1.1 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 does not 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.

8.7.1.2 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 XCP 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

8.7.1.3 Action

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

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

8.7.1.4 Error Severity

Error and Event messages are classified according to their Severity Level.

Table 203 XCP protocol 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.

8.7.2 Time-Out Handling

A Time-out Error occurs if the slave within a specified time does not 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 XCP Protocol supports 7 different Time-Out Values t_1 to t_7.
The master can get the values for t_1 to t_7 from the ASAM MCD-2 MC Description File.
The specific t_x for each command is indicated from table Table 205 Standard commands error handling up to Table 209 Non-volatile memory programming commands error handling.

8.7.2.1 Standard Communication Model

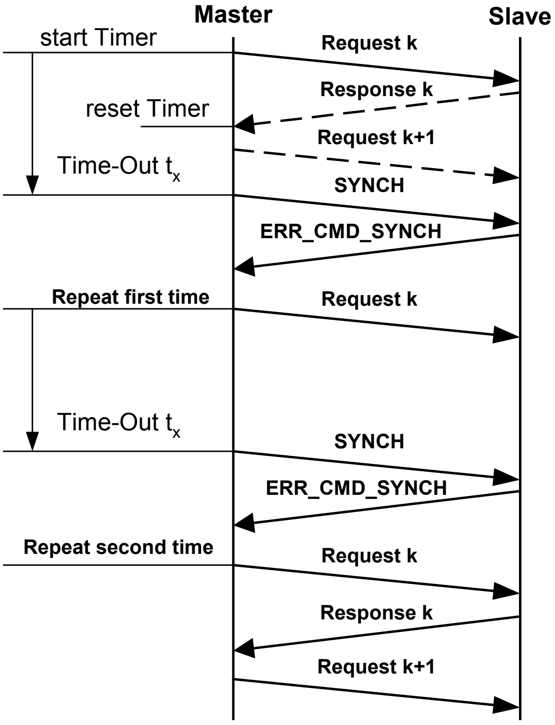

Figure 49 Time-out handling in standard communication model

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.

8.7.2.2 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.

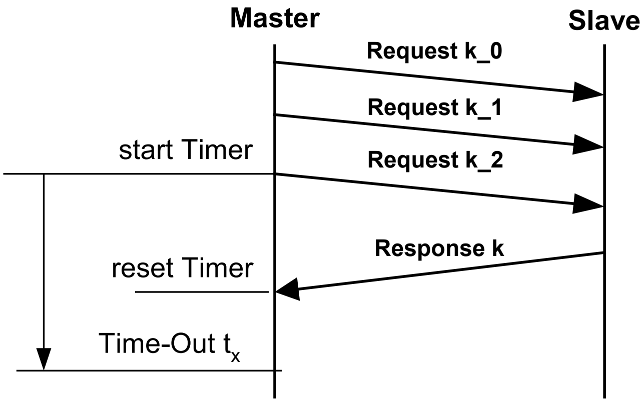

Figure 50 Time-out handling in master block transfer mode

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.

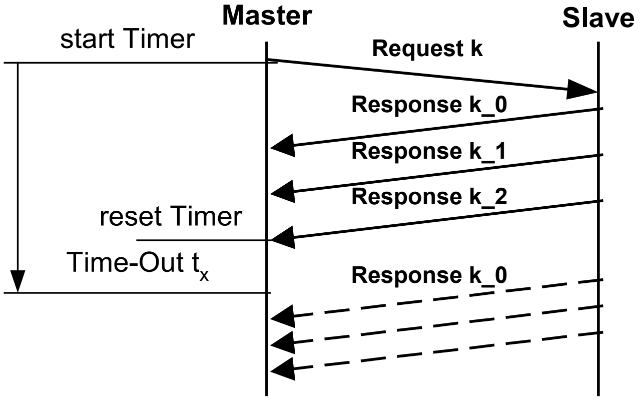

Figure 51 Time-out handling in slave block transfer mode

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.

8.7.2.3 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.

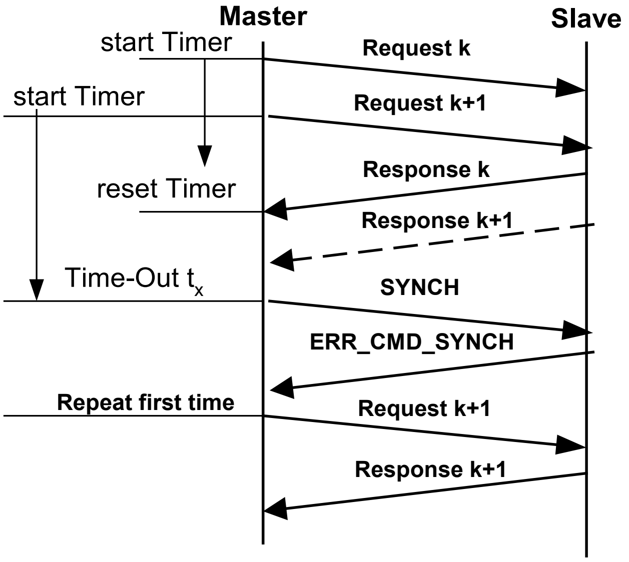

Figure 52 Time-out handling in interleaved communication model

8.7.2.4 Time-Out Manipulation

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

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 XCP commands are needed for this.

RESTARTING TIME-OUT DETECTION

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

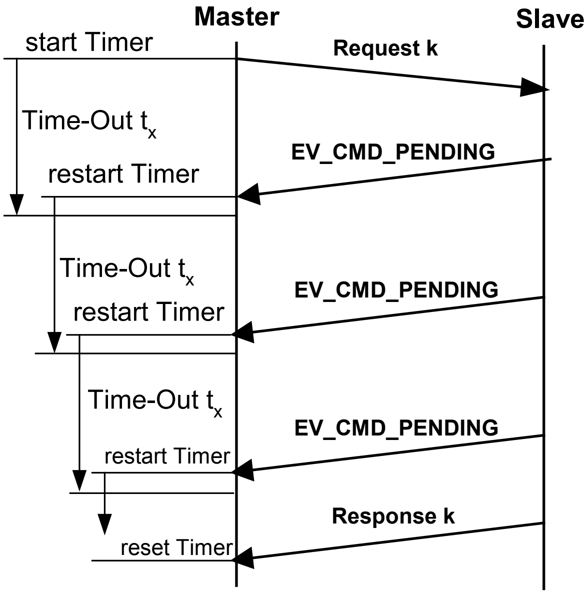

Figure 53 Restarting time-out detection with EV_CMD_PENDING

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.

8.7.3 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 perform the appropriate error handling (see Table 205, Table 206, Table 207, Table 208 and Table 209.
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 defined, the master has to check the Severity of this ERR in the Table 204 and decide about an appropriate reaction.

If an error occurs during a multi-command sequence, the master can decide about an appropriate reaction.
The Error packet codes in the table below can be sent as an Error packet with PID 0xFE as an answer to a CMD if the command has not been successfully executed.

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

Table 204 Error codes

Error	Code	Description	Severity
ERR_CMD_SYNCH	0x00	Command processor synchronization.	S0
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
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
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
ERR_RESOURCE TEMPORARY_NOT_ACCESSIBLE	0x33	Access to the requested resource is temporary not possible	S2

Table 205 Standard commands error handling

Command	Error	Pre-Action	Action
CONNECT(NORMAL)	timeout t_1	-	repeat ∞ times
	ERR_RES_TEMP_NOT_A.	display error	repeat
CONNECT(USER_DEFINED)	timeout t_6	wait t_7	repeat ∞ times
DISCONNECT	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
GET_STATUS	timeout t_1	SYNCH	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
SYNCH	timeout t_1	-	repeat 2 times
	ERR_CMD_SYNCH	-	–
	ERR_CMD_UNKNOWN	-	restart session
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_COMM_MODE_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_RES_TEMP_NOT_A.	-	skip
GET_ID	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_RES_TEMP_NOT_A.	-	skip
SET_REQUEST	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_SEED	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
UNLOCK	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error

Command	Error	Pre-Action	Action
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	-	restart session
	ERR_SEQUENCE	GET_SEED	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
SET_MTA	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
UPLOAD	timeout t_1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat
SHORT_UPLOAD	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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_RES_TEMP_NOT_A.	display error	repeat
BUILD_CHECKSUM	timeout t_2	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat

Command	Error	Pre-Action	Action
TRANSPORT_LAYER_CMD	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
USER_CMD	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat

Table 206 Calibration commands error handling

Command	Error	Pre-Action	Action
DOWNLOAD	timeout t_1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat
DOWNLOAD_NEXT	timeout t_1	SYNCH + DOWNLOAD	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat
DOWNLOAD_MAX	timeout t_1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times

Command	Error	Pre-Action	Action
	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_RES_TEMP_NOT_A.	display error	repeat
SHORT_DOWNLOAD	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat
MODIFY_BITS	timeout t_1	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat

Table 207 Page switching commands error handling

Command	Error	Pre-Action	Action
SET_CAL_PAGE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax

Command	Error	Pre-Action	Action
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_CAL_PAGE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_PAG_PROCESSOR_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
GET_SEGMENT_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	-	skip
GET_PAGE_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC

Command	Error	Pre-Action	Action
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	-	skip
SET_SEGMENT_MODE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_SEGMENT_MODE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
COPY_CAL_PAGE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_PAGE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat

Table 208 Data acquisition and stimulation commands error handling

Command	Error	Pre-Action	Action
SET_DAQ_PTR	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times

Command	Error	Pre-Action	Action
	ERR_DAQ_ACTIVE	-	repeat 2 times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
WRITE_DAQ	timeout t_1	SYNCH + SET_DAQ_PTR	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_PGM_ACTIVE	wait t_7	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
	ERR_RES_TEMP_NOT_A.	display error	repeat
	ERR_MEMORY_OVERFLOW	-	display error
SET_DAQ_LIST_MODE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
START_STOP_DAQ_LIST	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_MODE_NOT_VALID	-	retry other parameter

Command	Error	Pre-Action	Action
	ERR_DAQ_CONFIG	-	display error
	ERR_RES_TEMP_NOT_A.	display error	repeat
START_STOP_SYNCH	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_DAQ_CONFIG	-	display error
	ERR_RES_TEMP_NOT_A.	display error	repeat
CLEAR_DAQ_LIST	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	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_RES_TEMP_NOT_A.	display error	repeat
GET_DAQ_LIST_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
WRITE_DAQ_MULTIPLE	timeout t_1	SYNCH + SET_DAQ_PTR	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_PGM_ACTIVE	wait t_7	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

Command	Error	Pre-Action	Action
	ERR_MEMORY_OVERFLOW	-	display error
	ERR_DAQ_CONFIG	-	display error
	ERR_RES_TEMP_NOT_A.	display error	repeat
READ_DAQ	timeout t_1	SYNCH + SET_DAQ_PTR	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_DAQ_CLOCK	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
GET_DAQ_PROCESSOR_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
GET_DAQ_RESOLUTION_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
GET_DAQ_LIST_MODE	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax

Command	Error	Pre-Action	Action
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_DAQ_EVENT_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
FREE_DAQ	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
ALLOC_DAQ	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	reinit DAQ	repeat 2 times
	ERR_MEMORY_OVERFLOW	reinit DAQ	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
ALLOC_ODT	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times

Command	Error	Pre-Action	Action
	ERR_SEQUENCE	reinit DAQ	repeat 2 times
	ERR_MEMORY_OVERFLOW	reinit DAQ	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat
ALLOC_ODT_ENTRY	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	reinit DAQ	repeat 2 times
	ERR_MEMORY_OVERFLOW	reinit DAQ	retry other parameter
	ERR_RES_TEMP_NOT_A.	display error	repeat

Table 209 Non-volatile memory programming commands error handling

Command	Error	Pre-Action	Action
PROGRAM_START	timeout t_3	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DAQ_ACTIVE	START_STOP_x	repeat 2 times
	ERR_CMD_SYNTAX	–	retry other syntax
	ERR_ACCESS_LOCKED	unlock slave	repeat 2 times
	ERR_GENERIC	-	restart session
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM_CLEAR	timeout t_4	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	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_SEQUENCE	-	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM	timeout t_5	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_DENIED	-	display error

Command	Error	Pre-Action	Action
	ERR_ACCESS_LOCKED	unlock slave	repeat 2 times
	ERR_SEQUENCE	-	repeat 2 times
	ERR_MEMORY_OVERFLOW	-	display error
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM_RESET	timeout t_5	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_PGM_ACTIVE	-	repeat 2 times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	-	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
GET_PGM_PROCESSOR_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_RES_TEMP_NOT_A.	-	skip
GET_SECTOR_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use ASAM MCD-2 MC
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_MODE_NOT_VALID	-	retry other parameter
	ERR_SEGMENT_NOT_VALID	-	retry other parameter
	ERR_RES_TEMP_NOT_A.	-	skip
PROGRAM_PREPARE	timeout t_3	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	-	repeat 2 times
	ERR_GENERIC	-	restart session
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM_FORMAT	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error

Command	Error	Pre-Action	Action
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	-	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM_NEXT	timeout t_5	SYNCH + PROGRAM	repeat 2 times
	ERR_CMD_BUSY	wait t_7	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_MEMORY_OVERFLOW	-	display error
	ERR_SEQUENCE	-	repeat 2 times
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM_MAX	timeout t_5	SYNCH + SET_MTA	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	use alternative
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	-	repeat 2 times
	ERR_MEMORY_OVERFLOW	-	display error
	ERR_RES_TEMP_NOT_A.	display error	repeat
PROGRAM_VERIFY	timeout t_3	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	display error
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_ACCESS_LOCKED	Unlock slave	repeat 2 times
	ERR_SEQUENCE	-	repeat 2 times
	ERR_GENERIC	-	restart session
	ERR_VERIFY		new flashware version necessary
	ERR_RES_TEMP_NOT_A.	display error	repeat

8.8 Description of Events

The following chapters are a description of all possible XCP event packets.
Unused data bytes, marked as „reserved", may have arbitrary values.

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

8.8.1 Start in Resume Mode

Category	Event, optional
Mnemonic	EV_RESUME_MODE

Table 210 EV_RESUME_MODE event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x00
2	WORD	Session Configuration Id from slave
4	DWORD	Current slave Timestamp (optional)

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

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.

8.8.2 End of DAQ Clearing

Category	Event, optional
Mnemonic	EV_CLEAR_DAQ

Table 211 EV_CLEAR_DAQ event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x01

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

8.8.3 End of DAQ Storing

Category Event, optional
Mnemonic EV_STORE_DAQ

Table 212 EV_STORE_DAQ event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x02

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

8.8.4 End of CAL Storing

Category Event, optional
Mnemonic EV_STORE_CAL

Table 213 EV_STORE_CAL event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x03

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

8.8.5 Request to Restart Time-out Detection

Category Event, optional
Mnemonic EV_CMD_PENDING

Table 214 EV_CMD_PENDING event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x05

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

8.8.6 Indication of DAQ Overload

Category Event, optional
Mnemonic EV_DAQ_OVERLOAD

Table 215 EV_DAQ_OVERLOAD event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x06

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

8.8.7 Indication of Autonomous Disconnect

Category	Event, optional
Mnemonic	EV_SESSION_TERMINATED

Table 216 EV_SESSION_TERMINATED event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x07

With EV_SESSION_TERMINATED the slave indicates to the master that it autonomously decided to disconnect the current XCP session.

8.8.8 Transfer of Externally Triggered Timestamp

Category	Event, optional
Mnemonic	EV_TIME_SYNC

Table 217 EV_TIME_SYNC event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x08
2	BYTE	reserved
3	BYTE	reserved
4	DWORD	Timestamp

The generation of a timestamp can be triggered by an external sync line. This can be used for highly accurate time synchronization with the master without relying on the GET_DAQ_CLOCK mechanism.
The slave in this case with EV_TIME_SYNC sends its current time stamp to the master.
The returned timestamp has the format specified by the GET_DAQ_RESOLUTION_INFO command.
This event is not available if the slave does not support timestamps.

8.8.9 Indication of Time-out at STIM

Category	Event, optional
Mnemonic	EV_STIM_TIMEOUT

Table 218 EV_STIM_TIMEOUT event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x09
2	BYTE	Info Type 0 = Event channel number 1 = DAQ list number
3	BYTE	reserved
4	WORD	Event channel number or DAQ list number depending on Info Type.

If the slave detects a STIM timeout, it can notify the master by sending EV_STIM_TIMEOUT.
Info type defines whether the event channel or the DAQ list could not or just partially be stimulated.
The severity is implementation specific.

8.8.10 Entering Sleep Mode

Category	Event, optional
Mnemonic	EV_SLEEP

Table 219 EV_SLEEP event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x0A

With EV_SLEEP the slave indicates to the master that it enters SLEEP mode.

In SLEEP mode the slave stays in CONNECTED state but is not able of processing any commands. The slave will neither send any ERR_CMD_BUSY nor EV_CMD_PENDING.

If the master receives an EV_SLEEP, it must suspend sending commands until an EV_WAKE_UP is received.

Pending commands have to be discarded on both sides.

8.8.11 LEAVING SLEEP MODE

Category	Event, optional
Mnemonic	EV_WAKE_UP

Table 220 EV_WAKE_UP event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0x0B

With EV_WAKE_UP the slave indicates to the master that it leaves SLEEP mode and continues its normal operation.

8.8.12 USER-DEFINED EVENT

Category	Event, optional
Mnemonic	EV_USER

Table 221 EV_USER event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0xFE

EV_USER is a carrier for user-defined events.

8.8.13 TRANSPORT LAYER SPECIFIC EVENT

Category	Event, optional
Mnemonic	EV_TRANSPORT

Table 222 EV_TRANSPORT event packet

Position	Type	Description
0	BYTE	Event = 0xFD
1	BYTE	Event Code = 0xFF

EV_TRANSPORT is a carrier for Transport Layer specific events.

9 INTERFACE TO ASAM MCD-2 MC DESCRIPTION FILE

9.1 OVERVIEW

XCP consists of a generic Protocol Layer that can be transported on different Transport Layers.

The main.a2l that describes a slave supporting XCP on different Transport Layers, includes an **XCP_definitions.aml** that contains a reference to the Common_Parameters and a reference to the parameters that are specific for the different Transport Layers the slave supports. The generic protocol layer (see chapter 8) is independent from the used Transport layer. How the XCP protocol is transported by a particular Transport Layer like CAN, TCP/IP and UDP/IP is defined in the associated standards.

XCP_common_vX_Y.aml specifies the AML description of the Common_Parameters of the Protocol Layer.

XCP_on_##_vU_V.aml in the respective Associated standards ([6] [7] [8] [9] [10]) specifies the AML description of the specific parameters for each Transport Layer.

The main.a2l that describes a slave that supports XCP on different Transport Layers, also includes an **XCP_vX_Y.aml** that describes the structure of an “IF_DATA” for an XCP communication stack.
An “IF_DATA” for an XCP communication stack has the possibility to describe default Transport Layer independent parameters and Transport Layer specific parameters.
An “IF_DATA” for an XCP communication stack has the possibility to describe the overruling of the default parameters depending on the Transport Layer used.
Only an “IF_DATA XCPplus ..” has the possibility to describe multiple instances of one and the same Transport Layer.

The Compatibility Matrix gives an overview of the allowed combinations of Protocol Layer and Transport Layer parts.

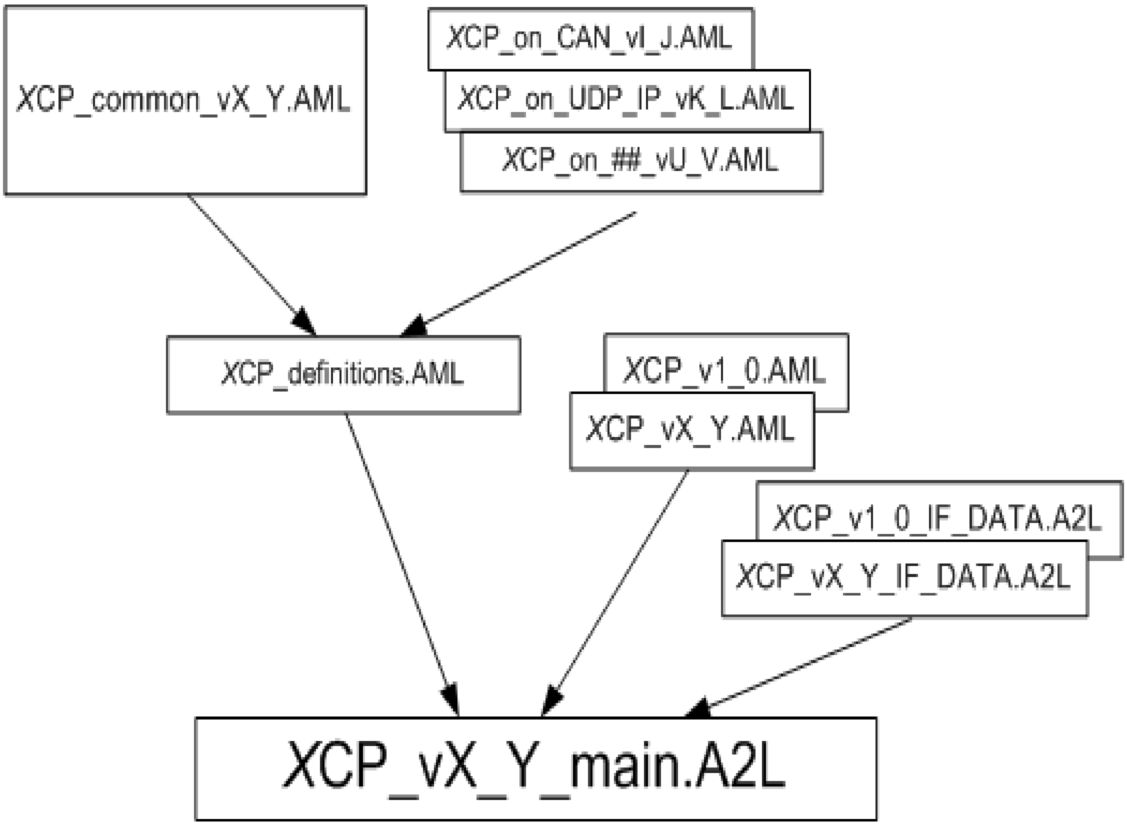

Figure 54 Structure of AML

9.2 ASAM MCD-2 MC AML FOR XCP (COMMON_PARAMETERS)

The following chapter describes the parameters that are independent from the Transport Layer used. They are grouped under the tag “Common_Parameters” in a file **XCP_common_vX_Y.aml** (vX_Y being the current XCP Protocol Layer Version Number).

```
/************************************************************** */
/*                                                              */
/* ASAP2 meta language for XCP protocol layer V1.2              */
/*                                                              */
/*                                                              */
/*   Datatypes:                                                 */
/*                                                              */
/*   A2ML       description                                     */
/*   ---------- ----------------------------------------------- */
/*   uchar      unsigned 8 Bit                                  */
/*   char       signed 8 Bit                                    */
/*   uint       unsigned integer 16 Bit                         */
/*   int        signed integer 16 Bit                           */
/*   ulong      unsigned integer 32 Bit                         */
/*   long       signed integer 32 Bit                           */
/*   float      float 32 Bit IEEE 745                           */
/*                                                              */
/****************************************************************/
```

```
/*************** start of PROTOCOL_LAYER *********************/
   struct Protocol_Layer {   /* At MODULE */

     uint;                           /* XCP protocol layer version */
                                     /* e.g. "1.2" = 0x0102        */
     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,
         "WRITE_DAQ_MULTIPLE"          = 0xC7

       })*;

       “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                  */

       "MAX_DTO_STIM" uint;
       /* overrules MAX_DTO see above for STIM use case */
     };

   }; /************ 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 */

    "DAQ_ALTERNATING_SUPPORTED" uint;
    /* Display_Event_Channel_Number */
    "PRESCALER_SUPPORTED";
    "RESUME_SUPPORTED";
    "STORE_DAQ_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";
    “MIN_ST_STIM” uchar; /* separation time between DTOs    */
                         /* time in units of 100 microseconds */
  };

};

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,
    “UNIT_1PS”   = 10,
    “UNIT_10PS”  = 11,
    “UNIT_100PS” = 12
  };

  taggedstruct {
    "TIMESTAMP_FIXED";
  };

};

"PID_OFF_SUPPORTED";
```

```
    /* Configuration Limits */
    "MAX_DAQ_TOTAL"         uint;
    "MAX_ODT_TOTAL"         uint;
    "MAX_ODT_DAQ_TOTAL"     uint;
    "MAX_ODT_STIM_TOTAL"    uint;
    "MAX_ODT_ENTRIES_TOTAL" uint;
    "MAX_ODT_ENTRIES_DAQ_TOTAL" uint;
    "MAX_ODT_ENTRIES_STIM_TOTAL" uint;


    "CPU_LOAD_MAX_TOTAL"    float;

    block "DAQ_MEMORY_CONSUMPTION" struct {
      ulong ; /* DAQ_MEMORY_LIMIT: in Elements[AG] */
      uint;   /* DAQ_SIZE: number of elements[AG] per DAQ list */
      uint;   /* ODT_SIZE: number of elements[AG] per ODT */
      uint;  /* ODT_ENTRY_SIZE: number of elements[AG] per ODT_entry
*/
      uint; /* ODT_DAQ_BUFFER_ELEMENT_SIZE: number of */
          /* payload elements[AG]*factor = sizeof(send buffer)[AG]*/
      uint; /* ODT_STIM_BUFFER_ELEMENT_SIZE: number of */
       /* payload elements[AG]*factor = sizeof(receive buffer)[AG]*/
    };
    /************ 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;            /* EVENT_CHANNEL_TIME_CYCLE     */
  uchar;            /* EVENT_CHANNEL_TIME_UNIT      */
  uchar;            /* EVENT_CHANNEL_PRIORITY       */

  taggedstruct  {   /* optional */
    “COMPLEMENTARY_BYPASS_EVENT_CHANNEL_NUMBER” uint;
    “CONSISTENCY” enum {
                        “DAQ”    = 0,
                        “EVENT” = 1
                        };
     block "MIN_CYCLE_TIME" struct {
          /* Configuration with 0-0 not allowed */
       uchar; /* EVENT_CHANNEL_TIME_CYCLE */
       uchar; /* EVENT_CHANNEL_TIME_UNIT  */
      };

     "CPU_LOAD_MAX" float;

     block "CPU_LOAD_CONSUMPTION_DAQ" struct {
       float;  /* DAQ_FACTOR */
       float;  /* ODT_FACTOR */
       float;  /* ODT_ENTRY_FACTOR */

       taggedstruct {
         (block "ODT_ENTRY_SIZE_FACTOR_TABLE" struct{
           uint; /* SIZE */
           float; /* SIZE_FACTOR */
         })*;
       };
     };

     block "CPU_LOAD_CONSUMPTION_STIM" struct {
       float;  /* DAQ_FACTOR */
       float;  /* ODT_FACTOR */
```

```
           float;  /* ODT_ENTRY_FACTOR */

           taggedstruct {
             (block "ODT_ENTRY_SIZE_FACTOR_TABLE" struct{

                  uint; /* SIZE */
                  float; /* SIZE_FACTOR */
               })*;
             };
           };

           block "CPU_LOAD_CONSUMPTION_QUEUE" struct {
             float;  /* ODT_FACTOR */
             float;  /* ODT_ELEMENT_LOAD,length in elements[AG] */
          };
       };

    })*;/************ 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 ************/
```

9.2.1 Protocol Layer and Transport Layer Parts (XCP_definitions.aml)

```
    /*************************************************************/
    /* XCP_definitions.aml has to include                        */
    /* a reference to a Protocol Layer part                      */
    /* and (a) reference(s) to that(those) Transport Layer(s)    */
    /* the slave supports                                        */
    /*                                                           */
    /* The Compatibility Matrix gives an overview of the allowed */
    /* combinations of Protocol Layer and Transport Layer parts  */
    /*                                                           */
    /*************************************************************/
    /********************** start of XCP definitions *****************/
    /include XCP_common_vX_Y.aml    /* protocol layer part           */
    /include XCP_on_##_vU_V.aml     /* transport layer part(s)       */
    /********************** end of XCP definitions *******************/
```

Example:

This slave supports XCP protocol version 1.0, when transported on UDP/IP in version 1.0 and when transported on CAN in version 1.1

```
    /********************** start of XCP definitions *****************/
    /include XCP_common_v1_0.aml     /* protocol layer part    */
    /include XCP_on_UDP_IP_v1_0.aml  /* transport layer UDP_IP */
    /include XCP_on_CAN_v1_1.aml     /* transport layer CAN    */
    /********************** end of XCP definitions *******************/
```

9.2.2 Combining the Parts to an XCP Communication Stack (XCP_vX_Y.aml)

The main.a2l that describes a slave that supports XCP on different Transport Layers, includes an XCP_vX_Y.aml that describes the structure of an “IF_DATA XCP ..” or of an “IF_DATA XCPplus ..”.
The structure of an “IF_DATA XCP ..” or “IF_DATA XCPplus ..” implies certain rules for combining a Protocol Layer part with one or more Transport Layer parts to build an XCP communication stack.
An “IF_DATA” for an XCP communication stack basically contains the Common_Parameters that are used as default values for communicating through XCP.
Inside at least one “/begin XCP_on_## ..” an “IF_DATA” for an XCP communication stack also contains specific parameters for a Transport Layer.
An “IF_DATA” for an XCP communication stack can contain references to different types of Transport Layers the slave supports.
An “IF_DATA XCP ..” cannot contain references to multiple instances of one and the same type of Transport Layer. In this case an “IF_DATA XCPplus ..” has to be used.
Inside a “/begin XCP_on_## ..” there exists the possibility to define Transport Layer specific values for the Common_Parameters that overrule the default Common_Parameters.
If looking for Common_Parameters for XCP on a specific Transport Layer, the master first has to check the availability of a Common_Parameters part inside the “/begin XCP_on_##” and use them if available. If this part is not available, the master has to use

the default values for the Common_Parameters as defined in the “IF_DATA XCP ..” or “IF_DATA XCPplus ..” respectively.

9.2.2.1 Structure of an IF_DATA “XCP”

```
/*************** start of XCP on different Transport Layers *******/
"XCP" struct {
  taggedstruct Common_Parameters ;          /* default parameters */
  taggedstruct {           /* transport layer specific parameters */
                      /* overruling of the default parameters     */
    block "XCP_ON_UDP_IP" struct {
      struct UDP_IP_Parameters ;           /* specific for UDP_IP */
      taggedstruct Common_Parameters;     /* overruling of default*/
    };
  };
};/**************** end of XCP on different Transport Layers ******/
```

9.2.2.2 Structure of an IF_DATA “XCPPLUS”

The main.a2l that describes a slave that supports XCP on different Transport Layers, should include an XCP_vX_Y.aml that describes the structure of an “IF_DATA XCPplus ..”.

The structure of an “IF_DATA XCPplus ..” implies the same rules for combining a Protocol Layer part with one or more Transport Layer parts to build an XCP communication stack, as the structure of an “IF_DATA XCP ..”.

Additionally, an “IF_DATA XCPplus ..” can contain references to multiple instances of one and the same type of Transport Layer.

If an “IF_DATA XCPplus ..” contains references to multiple instances of one and the same type of Transport Layer , the use of the tag “TRANSPORT_LAYER_INSTANCE” for indicating the different instances is mandatory.

```
/*****************************************************************/
/* XCP_vX_Y.aml always has to have the same structure            */
/* first there is a reference to the default parameters          */
/* then there is (a) reference(s) to that(those) Transport        */
/* Layer(s) your slave supports                                  */
/*                                                               */
/*****************************************************************/
/******** start of XCPplus on different Transport Layers ********/
"XCPplus" struct {
  uint;                              /* IF_DATA XCP version, use the
version of the standard, in this case 0x0102       */
  taggedstruct Common_Parameters ;  /* default parameters        */
  taggedstruct {          /* transport layer specific parameters */
                          /* overruling of the default parameters*/
   (block "XCP_ON_##" struct {
      struct ##_Parameters ;                     /* specific for */
      taggedstruct Common_Parameters;    /* overruling of default*/
      taggedstruct {       /* Identification of Transport Layer*/
          “TRANSPORT_LAYER_INSTANCE” char[101];
      };
    })*;
  };
};/********** end of XCPplus on different Transport Layers *******/
```

9.2.2.3 ASAM MCD-2 MC DESCRIPTION FILE CONTAINING AN IF_DATA “XCP” AND “XCPPLUS”

An ASAM MCD-2 MC description file can contain an “IF_DATA XCP ..” and an “IF_DATA XCPplus ..” at the same time.
If looking for communication parameters for an XCP stack, the master first has to check the availability of an “IF_DATA XCPplus ..” and apply the look-up rules as applicable for an “IF_DATA XCPplus ..”.
If this part is not available, the master has to check the availability of an “IF_DATA XCP ..”, and apply the look-up rules as applicable for an “IF_DATA XCP ..”.

9.3 EXAMPLE ASAM MCD-2 MC

9.3.1 EXAMPLE OF IF_DATA XCPPLUS (XCP_VX_Y_IF_DATA.A2L)

This chapter gives an example of an IF_DATA XCPplus at MODULE for a slave that supports XCP on UDP/IP and two instances of XCP on CAN.

For XCP on UDP/IP the default values for the Common_Parameters are used.
For the XCP on CAN instance identified as “private CAN” the DAQ part of the Common_Parameters is overruled. The XCP on CAN instance identified as “vehicle CAN” just contains other CAN specific parameters.

Example:

```
/begin IF_DATA XCPplus 0x0102  /* IF_DATA XCP version */

  /begin PROTOCOL_LAYER

    0x0102                            /* XCP protocol layer 1.2 */
    0x0019                             /* T1 [ms] */
    0x0019                             /* T2 [ms] */
    0x0019                             /* T3 [ms] */
    0x0019                             /* T4 [ms] */
    0x0019                             /* T5 [ms] */
    0x0005                             /* T6 [ms] */
    0x00C8                             /* T7 [ms] */
    0x20                               /* MAX_CTO */
    0x00FF                             /* MAX_DTO */

    BYTE_ORDER_MSB_FIRST
    ADDRESS_GRANULARITY_WORD
    SEED_AND_KEY_EXTERNAL_FUNCTION     "MyS&K.DLL"

    OPTIONAL_CMD  GET_ID
    OPTIONAL_CMD  SET_REQUEST
    OPTIONAL_CMD  GET_SEED
    OPTIONAL_CMD  UNLOCK
    OPTIONAL_CMD  SET_MTA
    OPTIONAL_CMD  UPLOAD
    OPTIONAL_CMD  BUILD_CHECKSUM
    OPTIONAL_CMD  DOWNLOAD
    OPTIONAL_CMD  SET_CAL_PAGE
    OPTIONAL_CMD  GET_CAL_PAGE
    OPTIONAL_CMD  COPY_CAL_PAGE
    OPTIONAL_CMD  CLEAR_DAQ_LIST
    OPTIONAL_CMD  SET_DAQ_PTR
    OPTIONAL_CMD  WRITE_DAQ
    OPTIONAL_CMD  SET_DAQ_LIST_MODE
    OPTIONAL_CMD  START_STOP_DAQ_LIST
    OPTIONAL_CMD  START_STOP_SYNCH
    OPTIONAL_CMD  GET_DAQ_CLOCK
    OPTIONAL_CMD  WRITE_DAQ_MULTIPLE

  /end PROTOCOL_LAYER

  /begin DAQ

    DYNAMIC                            /* DAQ_CONFIG_TYPE */
    0x0100                             /* MAX_DAQ */
    0x0100                             /* MAX_EVENT_CHANNEL */
    0x05                               /* MIN_DAQ */

    OPTIMISATION_TYPE_ODT_TYPE_32
    ADDRESS_EXTENSION_FREE
    IDENTIFICATION_FIELD_TYPE_RELATIVE_WORD_ALIGNED

    GRANULARITY_ODT_ENTRY_SIZE_DAQ_WORD
    0x04                         /* MAX_ODT_ENTRY_SIZE_DAQ */
```

```
NO_OVERLOAD_INDICATION
PRESCALER_SUPPORTED
RESUME_SUPPORTED

/begin STIM

  GRANULARITY_ODT_ENTRY_SIZE_STIM_WORD
  0x04                  /* MAX_ODT_ENTRY_SIZE_STIM */
  BIT_STIM_SUPPORTED

/end STIM

/begin TIMESTAMP_SUPPORTED

  0x0100                /* TIMESTAMP_TICKS */
  SIZE_WORD
  UNIT_1MS
  TIMESTAMP_FIXED

/end TIMESTAMP_SUPPORTED

/begin EVENT

  "10_ms_task"                 /* name */
  "10 ms"                      /* short name */
  0x0000                       /* EVENT_CHANNEL_NUMBER */
  DAQ_STIM
  0x02                         /* MAX_DAQ_LIST */
  0x0A                         /* EVENT_CHANNEL_TIME_CYCLE */
  0x06                         /* EVENT_CHANNEL_TIME_UNIT */
  0x00                         /* EVENT_CHANNEL_PRIORITY */

/end EVENT

/begin EVENT

  "100_ms_task"                /* name */
  "100 ms"                     /* short name */
  0x0001                       /* EVENT_CHANNEL_NUMBER */
  DAQ_STIM
  0x02                         /* MAX_DAQ_LIST */
  0x64                         /* EVENT_CHANNEL_TIME_CYCLE */
  0x06                         /* EVENT_CHANNEL_TIME_UNIT */
  0x10                         /* EVENT_CHANNEL_PRIORITY */
 CONSISTENCY EVENT

/end EVENT
```

```
        /begin CPU_LOAD_CONSUMPTION_DAQ
         1                       /* "DAQ_FACTOR" */
         2                       /* "ODT_FACTOR" */
         3                       /* "ODT_ENTRY_FACTOR" */
         /begin ODT_ENTRY_SIZE_FACTOR_TABLE
           1                     /* "SIZE" */
           150                   /* "SIZE_FACTOR", e.g. CPU cycles */
        /end ODT_ENTRY_SIZE_FACTOR_TABLE
        /begin ODT_ENTRY_SIZE_FACTOR_TABLE
           4                     /* "SIZE" */
           420                   /* "SIZE_FACTOR" */
        /end ODT_ENTRY_SIZE_FACTOR_TABLE
      /end CPU_LOAD_CONSUMPTION_DAQ

    /end DAQ

    /begin PAG

      0x01                          /* MAX_SEGMENTS */
      FREEZE_SUPPORTED

    /end PAG

    /begin PGM

      PGM_MODE_ABSOLUTE_AND_FUNCTIONAL
      0x02                          /* MAX_SECTORS */
      0x08                          /* MAX_CTO_PGM */

      /begin SECTOR

        "Lower sector"              /* name */
        0x00                        /* SECTOR_NUMBER */
        0x000000                    /* address */
        0x20000                     /* length */
        0x01                        /* Erase number */
        0x02                        /* Program number */
        0x00                        /* Programming method   */

      /end SECTOR

      /begin SECTOR

        "Upper sector"              /* name */
        0x01                        /* SECTOR_NUMBER */
        0x020000                    /* address */
        0x20000                     /* length */
        0x03                        /* Erase number */
        0x04                        /* Program number */
        0x00                        /* Programming method   */

      /end SECTOR

    /end PGM

    /begin XCP_ON_UDP_IP
```

```
         0x0100                   /* XCP on UDP_IP 1.0 */
         0x5555                   /* PORT           */
         ADDRESS "127.0.0.1"      /* ADDRESS   */

    /end XCP_ON_UDP_IP

    /begin XCP_ON_CAN

        0x0100                     /* XCP on CAN 1.0 */
        CAN_ID_BROADCAST  0x0100   /* auto-detection */
        CAN_ID_MASTER     0x0200   /* CMD/STIM */
        CAN_ID_SLAVE      0x0300   /* RES/ERR/EV/SERV/DAQ */
        BAUDRATE          500000   /* BAUDRATE */

        /begin DAQ_LIST_CAN_ID

           0x0000                     /* for DAQ_LIST 0 */
           FIXED 0x310

        /end DAQ_LIST_CAN_ID

        /begin DAQ_LIST_CAN_ID

           0x0001                     /* for DAQ_LIST 1 */
           FIXED 0x320

        /end DAQ_LIST_CAN_ID

        /begin DAQ_LIST_CAN_ID

           0x0002                     /* for DAQ_LIST 2 */
           FIXED 0x330

        /end DAQ_LIST_CAN_ID


        /begin PROTOCOL_LAYER

            0x0102                        /* XCP protocol layer 1.2 */
            0x000A                        /* T1 [ms] */
            0x000A                        /* T2 [ms] */
            0x000A                        /* T3 [ms] */
            0x000A                        /* T4 [ms] */
            0x000A                        /* T5 [ms] */
            0x0000                        /* T6 [ms] */
            0x0020                        /* T7 [ms] */
            0x08                          /* MAX_CTO */
            0x0008                        /* MAX_DTO */

            BYTE_ORDER_MSB_FIRST
            ADDRESS_GRANULARITY_BYTE

            OPTIONAL_CMD  SHORT_UPLOAD
            OPTIONAL_CMD  SHORT_DOWNLOAD
            OPTIONAL_CMD  DOWNLOAD_NEXT
```

```
          COMMUNICATION_MODE_SUPPORTED  BLOCK  SLAVE  MASTER  0x0A
0x02

      /end PROTOCOL_LAYER

    /begin DAQ

      STATIC                         /* DAQ_CONFIG_TYPE */
      0x0003                         /* MAX_DAQ */
      0x0002                         /* MAX_EVENT_CHANNEL */
      0x01                           /* MIN_DAQ */

      OPTIMISATION_TYPE_DEFAULT
      ADDRESS_EXTENSION_DAQ
      IDENTIFICATION_FIELD_TYPE_ABSOLUTE

      GRANULARITY_ODT_ENTRY_SIZE_DAQ_BYTE
      0x02                      /* MAX_ODT_ENTRY_SIZE_DAQ */

      OVERLOAD_INDICATION_EVENT
      PRESCALER_SUPPORTED
      RESUME_SUPPORTED

      /begin DAQ_LIST

        0x0000                       /* DAQ_LIST_NUMBER */
        DAQ_LIST_TYPE DAQ
        MAX_ODT                  0x01
        MAX_ODT_ENTRIES 0x02

        /begin PREDEFINED

             /begin ODT  0

                  ODT_ENTRY  0  0x4000  0x00  0x01  0xFF
                  ODT_ENTRY  1  0x4001  0x00  0x01  0xFF

             /end ODT

         /end PREDEFINED

      /end DAQ_LIST

      /begin DAQ_LIST

        0x0001                       /* DAQ_LIST_NUMBER */
        DAQ_LIST_TYPE DAQ_STIM
        MAX_ODT                  0x03
        MAX_ODT_ENTRIES 0x10

      /end DAQ_LIST

      /begin DAQ_LIST

        0x0002                       /* DAQ_LIST_NUMBER */
        DAQ_LIST_TYPE DAQ_STIM
        MAX_ODT                   0x10
```

```
            MAX_ODT_ENTRIES 0x20

          /end DAQ_LIST

          /begin EVENT

            "10_ms_task"                 /* name */
            "10 ms"                      /* short name */
            0x0000                       /* EVENT_CHANNEL_NUMBER */
            DAQ_STIM
            0x02                         /* MAX_DAQ_LIST */
            0x0A                         /* EVENT_CHANNEL_TIME_CYCLE */
            0x06                         /* EVENT_CHANNEL_TIME_UNIT */
            0x00                         /* EVENT_CHANNEL_PRIORITY */

          /end EVENT

          /begin EVENT

            "100_ms_task"                /* name */
            "100 ms"                     /* short name */
            0x0001                       /* EVENT_CHANNEL_NUMBER */
            DAQ_STIM
            0x02                         /* MAX_DAQ_LIST */
            0x64                         /* EVENT_CHANNEL_TIME_CYCLE */
            0x06                         /* EVENT_CHANNEL_TIME_UNIT */
            0x10                         /* EVENT_CHANNEL_PRIORITY */

          /end EVENT

        /end DAQ

     TRANSPORT_LAYER_INSTANCE “private CAN”

      /end XCP_ON_CAN

      /begin XCP_ON_CAN

          0x0100                     /* XCP on CAN 1.0 */
          CAN_ID_BROADCAST  0x0100   /* auto-detection */
          CAN_ID_MASTER     0x0400   /* CMD/STIM */
          CAN_ID_SLAVE      0x0500   /* RES/ERR/EV/SERV/DAQ */
          BAUDRATE          500000   /* BAUDRATE */

     TRANSPORT_LAYER_INSTANCE “vehicle CAN”

      /end XCP_ON_CAN

    /end IF_DATA
```

9.3.2 EXAMPLE OF MAIN *.A2L FILE (XCP_VX_Y_MAIN.A2L)

This chapter gives an example of an ASAM MCD-2 MC description file for a slave that supports XCP on UDP/IP and XCP on CAN.

9.3.2.1 Example of Main *.A2L file containing an IF_DATA “XCPplus”

```
/begin PROJECT XCP
  "XCP on different Transport Layers"

  /begin HEADER
   "Example of multiple instances principle"

    VERSION    "Sue01"
    PROJECT_NO XCPv01

  /end HEADER

  /begin MODULE XCP_Sim
    "Simulator by Vector Informatik GmbH"

    /begin A2ML

      /include XCP_definitions.aml

        block "IF_DATA" taggedunion if_data {

        /include XCP_v1.2.aml

      };

    /end A2ML

    /begin MOD_COMMON ""

      BYTE_ORDER MSB_LAST

    /end MOD_COMMON

    /include XCP_v1_2_IF_DATA.a2l

    /begin MOD_PAR “”

      /begin MEMORY_SEGMENT

        Calib                        /* name */
        "Calibration data"           /* long identifier */
        DATA                         /* PrgType */
        FLASH                        /* Memory Type */
        INTERN                       /* Attribute */
        0x4000                       /* Address */
        0x200                        /* Size */
        -1 -1 -1 -1 -1               /* no mirrored segments */

        /begin IF_DATA XCPplus  0x0102   /* IF_DATA XCP version */

          /begin SEGMENT

            0x00                      /* segment logical number */
            0x02                      /* number of pages */
            0x00                      /* address extension */
            0x00                      /* Compression method   */
            0x00                      /* Encryption method      */

            /begin CHECKSUM

              XCP_USER_DEFINED   /*    checksum    through    external
function*/
              MAX_BLOCK_SIZE  0x100    /*  maximum block size */
              EXTERNAL_FUNCTION   "MyChecksum.DLL"      /*    name  of
function */

            /end CHECKSUM

            /begin PAGE

              0x00                    /* page number */
              ECU_ACCESS_DONT_CARE
              XCP_READ_ACCESS_DONT_CARE
              XCP_WRITE_ACCESS_NOT_ALLOWED
              INIT_SEGMENT 0x00       /* init segment */

            /end PAGE

            /begin PAGE

              0x01                    /* page number */
              ECU_ACCESS_DONT_CARE
              XCP_READ_ACCESS_DONT_CARE
              XCP_WRITE_ACCESS_WITH_ECU_ONLY
              INIT_SEGMENT 0x00       /* init segment */

            /end PAGE

            /begin ADDRESS_MAPPING

              0x04000                 /* from */
              0x14000                 /* to */
              0x100                   /* length */

            /end ADDRESS_MAPPING

            /begin ADDRESS_MAPPING

              0x04100                 /* from */
              0x24100                 /* to */
              0x100                   /* length */

            /end ADDRESS_MAPPING

          /end SEGMENT

        /end IF_DATA

      /end MEMORY_SEGMENT

    /end MOD_PAR
    /begin MEASUREMENT

      Triangle                /* name                */
      "Triangle test signal"  /* long identifier    */

      SBYTE                   /* DataType            */
      BitSlice.CONVERSION     /* conversion          */
      0                       /* resolution          */
      0                       /* accuracy            */
      -50  50                 /* lower, upper limit */

      BIT_MASK 0xFF
      ECU_ADDRESS 0x44A16
      FORMAT "%7.3"

      /begin IF_DATA XCPplus  0x0102  /* IF_DATA XCP version */

        /begin DAQ_EVENT VARIABLE

            /begin AVAILABLE_EVENT_LIST
                EVENT 0001  EVENT 0002
            /end AVAILABLE_EVENT_LIST

            /begin DEFAULT_EVENT_LIST
                EVENT 0001
            /end DEFAULT_EVENT_LIST

        /end DAQ_EVENT

      /end IF_DATA

    /end MEASUREMENT

    /begin COMPU_METHOD

      BitSlice.CONVERSION
      ""
      RAT_FUNC
      "%2.0"
      "-"
      COEFFS 0 1 0 0 0 1

    /end COMPU_METHOD

  /end MODULE

/end PROJECT
```

9.4 Consistency Between ASAM MCD-2 MC and Slave

The parameterization of the XCP protocol can be described in IF_DATA sections of an ASAM MCD-2 MC description file.
If supported, the master also can read out almost all of these parameters directly from the slave.

If for a parameter there is both information in the ASAM MCD-2 MC file and by reading it out from the slave, the master has to check the consistency of both values.
If the master detects an inconsistency, he has to inform the user about the detected inconsistency. The master has to give the user the possibility to decide whether the master for this parameter has to use the value from the ASAM MCD-2 MC description file or the value read from the slave.

10 INTERFACE TO AN EXTERNAL SEED&KEY FUNCTION

When calculating a Key from a Seed, the Master always has to use a user-defined algorithm. This algorithm is provided by the slave vendor. It contains functions to read out the provided privileges and to calculate a Key from a Seed.

The "SEED_AND_KEY_EXTERNAL_FUNCTION" parameter at the "PROTOCOL_LAYER" section in the ASAM MCD-2 MC Description File, indicates the Name of the external function file the Master has to use. The parameter is an ASCII string that contains the name and the extension but does not contain the path to the file.

The integration of this function file is programming language and platform dependent. E.g. when using a Windows ® operating system, these "external functions" could be located in a MySeedNKey.DLL (Dynamically Linked Library). When using a UNIX ® operating system, these "external functions" could be located in a MySeedNKey.SO (Shared Object).

The mechanism required to include external functions files is tool specific.
However, the included functions and calling parameters themselves are specified in this chapter.

To have an easy handling for XCP there is only one external function file which may contain all algorithms to unlock all privileges or only a subset. That means the supplier can generate different external function files with different privilege level combinations.

The privilege levels are described based on the "Resource Mask" of XCP and coded as defined there.
The ECU needs one algorithm for each privilege (if protected).

The external function file contains 2 functions: one to get information about the available privileges of this function file and one to calculate a key from a seed for the requested privilege.

10.1 FUNCTION XCP_GETAVAILABLEPRIVILEGES

Table 223 XCP_GetAvailablePrivileges parameters

Parameter name:	Data type	XCP_ComputeKeyFromSeed	Remarks
Return Value:	DWORD	Error Code	
Parameter 1:	BYTE *	Available Privilege	returns the privileges with available unlock algorithms in this external function file

Function returns available privileges as XCP Resource Availability Mask.

The following error codes can be returned: XcpSkExtFncAck: o.k.

If the master, by using an external function on an Intel-based platform, calculates a Key from a Seed for an ECU running a Motorola format, it is not in the responsibility of the master to adjust the byte order. The external function receives and returns BYTE arrays in exactly the order as transmitted in the XCP messages.

10.2 Function XCP_ComputeKeyFromSeed

Table 224 XCP_ComputeKeyFromSeed parameters

Parameter name:	Data type	XCP_ComputeKeyFromSeed	Remarks
Return Value:	DWORD	Error Code	
Parameter 1:	BYTE	Requested Privilege	=> from Tool,
			- input for external function
			- input for GetSeed command
Parameter 2:	BYTE	Byte Length Seed	from answer of GetSeed
Parameter 3:	BYTE *	Pointer to Seed	
Parameter 4:	BYTE *	Byte Length Key	
			input: max bytes memory for key
			output: byte length of key
Parameter 5:	BYTE *	Pointer to Key	

The external function `XCP_ComputeKeyFromSeed` should calculate Key from Seed for the requested privilege

Key = f(Seed, RequestedPrivilege) (only one privilege can be unlocked at once)

Remark:
Parameter 4 “Byte Length Key” must be initialised with the maximum Length of Key reserved by the Master when calling the external Seed&Key function. This makes sure that the Seed&Key function will not write into other memory than reserved. It is recommended to reserve 255 bytes since this is the maximum length that is possible.

The following error codes can be returned:

- XcpSkExtFncAck: = 0 o.k.
- XcpSkExtFncErrPrivilegeNotAvailable = 1 the requested privilege cannot be unlocked with this function
- XcpSkExtFncErrInvalidSeedLength = 2 the seed length is wrong, key could not be computed
- XcpSkExtFncErrUnsufficientKeyLength = 3 the space for the key is too small

Example:

Example source code for a Windows ® -DLL are distributed together with this specification (SeedNKeyXCP.*).

11 INTERFACE TO AN EXTERNAL CHECKSUM FUNCTION

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 external function.

The integration of this function file is programming language and platform dependent. E.g. when using a Windows ® operating system, this “external function” could be located in a MyChecksum.DLL (Dynamically Linked Library). When using a UNIX ® operating system, this “external function” could be located in a MyChecksum.SO (Shared Object).

The mechanism required to include external functions files is tool specific.
However, the included function and calling parameters themselves are specified in this chapter.

The “EXTERNAL_FUNCTION” parameter at the “CHECKSUM” block at an XCP SEGMENT in the ASAM MCD-2 MC Description File, indicates the Name of the external function file the Master has to use. The parameter is an ASCII string that contains the name and the extension but does not contain the path to the file.

The API for calling a Win32 Checksum.DLL is described in [11].

12 INTERFACE TO AN EXTERNAL A2L DECOMPRESSION/DECRYPTING FUNCTION

When an XCP slave returns the A2L description data in a compressed and/or encrypted format, the XCP master has to pass it to an external function which is responsible for decompression and/or decrypting and is provided by the slave vendor.
The integration of this function file is programming language and platform dependent.
The mechanism required to include external function files is tool specific.
However, the included functions and calling parameters themselves are specified below.

Function prototype:

```
int XCP_DecompressA2L(
unsigned int compressedLength,       // IN: the length in bytes of the compressed/encrypted data block
unsigned char* compressedData,       // IN: the pointer to the start of the compressed/encrypted data block
unsigned int* decompressedLength,    // OUT: a pointer to a location where the function saves the
                                     // decompressed block size
unsigned char**                      // OUT: a pointer to the location where the function saves the
decompressedData);                   // decompressed data pointer
```

Return values:

- 0 = successful execution
- 1 = corrupt source data
- 2 = not enough memory for decompressed/decrypted data
- 3 = internal error (should not be used normally)
- 4 = SmartCard not accessible

Description:

The function allocates the memory for the decompressed/decrypted data itself. The client code can use the data after successful execution

The client code is responsible for releasing the decompressed/decrypted memory block by calling the following function.

Function prototype:

```
int XCP_ReleaseDecompressedData (unsigned char* decompressedData);
```

Return values:

- 0 = successful execution
- 1 = internal error, buffer is not released

Description:

After executing this function, the decompressed memory block must not be accessed anymore.

13 Examples

13.1 Configuration Examples

Table 225 CPU load calculation examples

ODT_ENTRY_SIZE	SIZE(1)	SIZE(2)	SIZE(4)	SIZE(5)	Calculation	Result
1	2	3	5	-	1 * 2	2
3	2	3	5	-	2 * 3	6
4	2	3	5	-	1 * 5	5
5	2	3	5	-	2 * 5	10
15	2	3	5	-	4 * 5	20
253	2	3	5	-	64*5	320
253	2	3	-	-	127*3	381
253	2	-	-	-	253*2	506
253	2	3	5	10	51*10	510
253	2	3	5	7.5	51*7.5	382.5
253	2	3	5	6.25	51*6.25	318.75

13.2 Examples for GET_ID Identification Strings

Table 226 GET_ID identification types

Identification type	String
1	Test
2	c:\database\test.a2l
3	ftp://ttp.oem.com\data_repository\project_xcp\test.a2l

13.3 Example Communication Sequences

The sequences below are supplied to aid the understanding of the relationship between individual commands.

Table 227 Notation for indicating the packet direction

Symbol	Direction	Packet direction
→	CMD	Master to Slave
←	RES	Slave to Master

13.4 SETTING UP A SESSION

Table 228 Getting BASIC information

Direction	XCP Packet	Parameters
CONNECT		
→	FF 00	mode= 0x00
		=> NORMAL
←	FF 15 C0 08 08 00 10 10	RESOURCE=0x15
		=> CAL/PAG, DAQ, PGM available
		COMM_MODE_BASIC=0xC0
		=> Byte Order = Intel
		Address_Granularity = Byte
		Slave Block Mode available
		GET_COMM_MOD_INFO provides additional information
		MAX_CTO = 0x08
		MAX_DTO = 0x0008
		XCP Protocol Layer Version = 0x10
		XCP Transport Layer Version = 0x10
GET_COMM_MODE_INFO		
→	FB	
←	FF xx 01 xx 02 00 xx 64	COMM_MODE_OPTIONAL=0x01
		=> Master Block Mode available
		MAX_BS = 0x02
		MIN_ST = 0x00
		XCP Driver Version = 0x64
GET_STATUS		
→	FD	
←	FF 00 15 xx 00 00	Current Session Status = 0x00
		=> no request active,
		Resume not active,
		no DAQ running
		Resource Protection Status = 0x15
		=> CAL/PAG, DAQ, PGM are protected
		Session Configuration ID= 0x0000
		=> no RESUME session configured

Table 229 Unlocking protected resources through a Seed&Key Mechanism

Direction	XCP Packet	Parameters
GET_SEED		
→	F8 00 01	Mode = 0x00
		=> first part of seed
		resource = 0x01
		=> CAL/PAG to be unlocked
←	FF 06 00 01 02 03 04 05	Mode = 0x00
		=> total length of seed = 0x06
		Seed = 0x00 0x01 0x02 0x03 0x04 0x05
UNLOCK		
→	F7 06 69 AB A6 00 00 00	Length of key = 0x06
		Key = 0x69 0xAB 0xA6 0x00 0x00 0x00
←	FF 14	Current Protection Status = 0x14
		=> CAL/PAG unlocked,
		DAQ still protected,
		PGM still protected
GET_SEED		
→	F8 00 04	Mode = 0x00
		=> first part of seed
		resource = 0x04
		=> DAQ to be unlocked
←	FF 06 06 07 08 09 0A 0B	Mode = 0x00
		=> total length of seed = 0x06
		Seed = 0x06 0x07 0x08 0x09 0x0A 0x0B
UNLOCK		
→	F7 06 96 BA 6A 00 00 00	Length of key = 0x06
		Key = 0x96 0xBA 0x6A 0x00 0x00 0x00
←	FF 10	Current Protection Status = 0x10
		=> CAL/PAG unlocked,
		DAQ unlocked,
		PGM still protected
GET_SEED		
→	F8 00 10	Mode = 0x00
		=> first part of seed
		resource = 0x10
		=> PGM to be unlocked
←	FF 06 05 04 03 02 01 00	Mode = 0x00
		=> total length of seed = 0x06

Direction	XCP Packet	Parameters
		Seed = 0x05 0x04 0x03 0x02 0x01 0x00
UNLOCK		
→	F7 06 11 22 33 22 11 00	Length of key = 0x06
		Key = 0x11 0x22 0x33 0x22 0x11 0x00
←	FF 00	Current Protection Status = 0x00
		=> CAL/PAG unlocked,
		DAQ unlocked,
		PGM unlcoked

Table 230 Getting information about the slave's description file

Direction	XCP Packet	Parameters
GET_ID		
→	FA 01	Requested Identification Type = 0x01
		=> ASAM MC 2 filename without path and extension
←	FF 00 xx xx 06 00 00 00	Mode = 0x00
		=> MTA set automatically, UPLOAD needed
		Length = 0x00000006
UPLOAD		
→	F5 06	Number of data elements = 0x06
←	FF 58 43 50 53 49 4D	Data elements in ASCII
		=> 58 43 50 53 49 4D
		X C P S I M

13.5 CALIBRATING

For n = 0 to MAX_SEGMENTS-1 do

Table 231 Getting the current active pages for ECU access

Direction	XCP Packet	Parameters
GET_CAL_PAGE		
→	EA 01 00	Access mode = 0x01
		=> ECU access
		SEGMENT_NUMBER = 0x00 (= n)
←	FF xx xx 01	Current active page = 0x01

For n = 0 to MAX_SEGMENTS-1 do

Table 232 Getting the current active pages for XCP master access

Direction	XCP Packet	Parameters
GET_CAL_PAGE		
→	EA 02 00	Access mode = 0x02
		=> XCP master access
		SEGMENT_NUMBER = 0x00 (= n)
←	FF xx xx 01	Current active page = 0x01

Table 233 Equalizing master and slave through checksum calculation

Direction	XCP Packet	Parameters
SET_CAL_PAGE		
→	EB 83 xx 00	mode= 0x83
		=> ECU access and XCP access,
		for all segments (segment number ignored)
		Page Number = 0x00
←	FF	
SET_MTA		
→	F6 xx xx 00 3C 00 00 00	Address extension = 0x00
		Address = 0x0000003C
←	FF	
BUILD_CHECKSUM		
→	F3 xx xx xx AD 0D 00 00	Block size = 0x00000DAD
←	FF 02 xx xx 2C 87 00 00	Checksum type = 0x02
		=> XCP_ADD_12, byte into word
		Checksum = 0x0000872C

Table 234 Reading/writing slave parameters

Direction	XCP Packet	Parameters
SET_MTA		
→	F6 xx xx 00 60 00 00 00	Address extension = 0x00
		Address = 0x00000060
←	FF	
DOWNLOAD		
→	F0 04 00 00 80 3F	Number of data elements = 0x04
		Data elements = 0x00 0x00 0x80 0x3F
←	FF	

Direction	XCP Packet	Parameters
SHORT_UPLOAD		
→	F4 04 xx 00 60 00 00 00	Number of data elements = 0x04
		Address extension = 0x00
		Address = 0x00000060
←	FF 00 00 80 3F	Data elements = 0x00 0x00 0x80 0x3F

Table 235 Copying between pages

Direction	XCP Packet	Parameters
COPY_CAL_PAGE		
→	E4 00 01 02 03	Source Segment Number = 0x00
		Source Page Number = 0x01
		Destination Segment Number = 0x02
		Destination Page Number = 0x03
←	FF	

13.6 Synchronous Data Transfer

13.6.1 Getting Information About the Slave's DAQ List Processor

Table 236 Getting information about the slave's DAQ list processor

Direction	XCP Packet	Parameters
GET_DAQ_PROCESSOR_INFO		
→	DA	
←	FF 11 00 00 01 00 00 40	DAQ_PROPERTIES = 0x11
		=> DAQ_config_type = dynamic,
		timestamp_supported
		MAX_DAQ = 0x0000 (dynamic)
		MAX_EVENT_CHANNEL = 0x0001
		MIN_DAQ = 0x00, no predefined lists
		DAQ_KEY_BYTE = 0x40
		=> Optimisation_default,
		address extension free,
		Identification_field_type "rel. ODT+DAQ(BYTE)"
GET_DAQ_RESOLUTION_INFO		
→	D9	
←	FF 02 FD xx xx 62 0A 00	Granularity_odt_entry_size_daq = 0x02
		Max_odt_entry_size_daq = 0xFD
		Timestamp_mode = 0x62
		=> size = WORD,
		unit = 1 ms
		Timestamp_ticks = 0x000A

For n = 0 to MAX_EVENT_CHANNEL-1 do

Table 237 Getting information about EVENTS

Direction	XCP Packet	Parameters
		GET_DAQ_EVENT_INFO
→	D7 xx 00 00	Event_channel_number = 0x0000 (= n)
←	FF 04 01 05 0A 60 00	DAQ_EVENT_PROPERTIES = 0x04
		=> Event_channel_type = DAQ
		MAX_DAQ_LIST = 0x01
		Event channel name length = 0x05
		Event channel time cycle = 0x0A
		Event channel time unit = 0x60
		=> 1 ms
		Event channel priority = 0x00
		=> lowest
		UPLOAD
→	F5 05	Number of data elements = 0x05
←	FF 31 30 20 6D 73	Data elements in ASCII
		=> 31 30 20 6D 73
		1 0 m s

For a slave with DAQ_config_type = static, the response on GET_DAQ_PROCESSOR_INFO could look like:

FF 10 01 00 01 00 00 40

Additionally to GET_DAQ_RESOLUTION_INFO and the loop with (GET_DAQ_EVENT_INFO + UPLOAD), for a slave with DAQ_config_type = static it makes sense to get the information about the statically allocated DAQ lists:

For n = 0 to MAX_DAQ-1 do

Table 238 Getting information about DAQ lists

Direction	XCP Packet	Parameters
		GET_DAQ_LIST_INFO
→	D8 xx 00 00	DAQ_list_number = 0x0000
←	FF 04 03 0A	DAQ_LIST_PROPERTIES = 0x04
		=> DAQ_list_type = DAQ only
		MAX_ODT = 0x03
		MAX_ODT_ENTRIES = 0x0A

13.6.2 Preparing the DAQ Lists

13.6.2.1 Static Configuration

For n = MIN_DAQ to MAX_DAQ-1 do

Table 239 Clearing static DAQ lists

Direction	XCP Packet	Parameters
CLEAR_DAQ_LIST		
→	E3 xx 00 00	DAQ_LIST_NUMBER = 0x0000
←	FF	

13.6.2.2 Dynamic Configuration

Table 240 Dynamic DAQ list configuration

Direction	XCP Packet	Parameters
FREE_DAQ		
→	D6	
←	FF	
ALLOC_DAQ		
→	D5 xx 01 00	DAQ_COUNT = 0x0001
←	FF	

For n = MIN_DAQ to MIN_DAQ+DAQ_COUNT-1 do

Table 241 Dynamic ODT allocation

Direction	XCP Packet	Parameters
ALLOC_ODT		
→	D4 xx 00 00 01	DAQ_LIST_NUMBER = 0x0000 (= n)
		ODT_COUNT = 0x01
←	FF	

For n = MIN_DAQ to MIN_DAQ+DAQ_COUNT-1 do
For i = 0 to ODT_COUNT(n)-1 do

Table 242 Dynamic ODT entry allocation

Direction	XCP Packet	Parameters
ALLOC_ODT_ENTRY		
→	D3 xx 00 00 00 02	DAQ_LIST_NUMBER = 0x0000 (= n)
		ODT_NUMBER = 0x00 (= i)
		ODT_ENTRIES_COUNT = 0x02
←	FF	

13.6.3 Configuring the DAQ Lists

For n = MIN_DAQ to N_Upper_Limit do
For i = 0 to I_Upper_Limit do

Table 243 Addressing an ODT entry

Direction	XCP Packet	Parameters
SET_DAQ_PTR		
→	E2 xx 00 00 00 00	DAQ_LIST_NUMBER = 0x0000 (= n)
		ODT_NUMBER = 0x00 (= i)
		ODT_ENTRY_NUMBER = 0x00
←	FF	

For j = 0 to J_Upper_Limit do

Table 244 Configuration of an ODT entry

Direction	XCP Packet	Parameters
WRITE_DAQ		
→	E1 FF 04 00 08 55 0C 00	BIT_OFFSET = 0xFF
		=> normal data element
		Size of element = 0x04
		Address extension = 0x00
		Address = 0x000C5508
←	FF	

For the loops the following applies:

Table 245 Loop ranges

DAQ_CONFIG_TYPE	Static	Dynamic
N_Upper_Limit	MAX_DAQ-1	MIN_DAQ+DAQ_COUNT-1
I_Upper_Limit	MAX_ODT(n)-1	ODT_COUNT(n)-1
J_Upper_Limit	MAX_ODT_ENTRIES(n,i)-1	ODT_ENTRIES_COUNT(n,i)-1

13.6.4 Starting the Data Transfer

For n = 0 to MAX_DAQ-1 do

Table 246 Configuration of DAQ list mode

Direction	XCP Packet	Parameters
	SET_DAQ_LIST_MODE	
→	E0 10 00 00 00 00 01 00	Mode = 0x10
		=> DIRECTION = DAQ,
		timestamped
		DAQ_LIST_NUMBER = 0x0000 (= n)
		EVENT_CHANNEL_NUMBER = 0x0000
		Prescaler = 01
		=> no reduction
		DAQ list priority = 00
		=> lowest
←	FF	

For n = 0 to MAX_DAQ-1 do

Table 247 Preparing the data acquisition for DAQ lists

Direction	XCP Packet	Parameters
	START_STOP_DAQ_LIST	
→	DE 02 00 00	Mode = 0x02
		=> select
		DAQ_LIST_NUMBER = 0x0000 (= n)
←	FF	

Table 248 Time synchronization and start of data acquisition

Direction	XCP Packet	Parameters
	GET_DAQ_CLOCK	
→	DC	
←	FF xx xx xx AA C5 00 00	Receive timestamp = 0x0000C5AA
	START_STOP_SYNCH	
→	DD 01	Mode = 0x01
		=> start selected
←	FF	

13.6.5 Stopping the Data Transfer

For n = 0 to MAX_DAQ-1 do

Table 249 Preparing the stop of data acquisition

Direction	XCP Packet	Parameters
START_STOP_DAQ_LIST		
→	DE 02 00 00	Mode = 0x02
		=> select
		DAQ_LIST_NUMBER = 0x0000 (= n)
←	FF	

Table 250 Stopping the data acquisition

Direction	XCP Packet	Parameters
START_STOP_SYNCH		
→	DD 02	Mode = 0x02
		=> stop selected
←	FF	

13.7 Reprogramming the Slave

Table 251 Indicating the beginning of a programming sequence

Direction	XCP Packet	Parameters
PROGRAM_START		
→	D2	
←	FF xx 01 08 2A FF	COMM_MODE_PGM = 0x01
		=> Master Block Mode supported
		MAX_CTO_PGM = 0x08
		MAX_BS_PGM = 0x2A
		MIN_ST_PGM = 0xFF

Table 252 Clearing a part of non-volatile memory

Direction	XCP Packet	Parameters
SET_MTA		
→	F6 xx xx 00 00 01 00 00	Address extension = 0x00
		Address = 0x00000100
←	FF	
PROGRAM_CLEAR		
→	D1 00 xx xx 00 01 00 00	mode= 0x00
		=> Absolute access mode
		Clear range = 0x00000100
←	FF	

Table 253 Selecting a non-volatile memory segment

Direction	XCP Packet	Parameters
SET_MTA		
→	F6 xx xx 00 00 01 00 00	Address extension = 0x00
		Address = 0x00000100
←	FF	

Loop with PROGRAM until end of SEGMENT

Table 254 Programming data to a non-volatile memory segment

Direction	XCP Packet	Parameters
PROGRAM		
→	D0 06 00 01 02 03 04 05	Size = 0x06
		Data elements = 0x00 0x01 0x02 0x03 0x04 0x05
←	FF	

Table 255 Indicating the end of a programming sequence

Direction	XCP Packet	Parameters
PROGRAM_RESET		
→	CF	
←	FF	

13.8 Closing a Session

Table 256 Closing a session

Direction	XCP Packet	Parameters
DISCONNECT		
→	FE	
←	FF	

Figure Directory

Table Directory

Bibliography

[1] **ASAM MCD-2 MC**/"Measurement and Calibration Data Specification", Version 1.6.x

[2] **ISO 14229-1**/Road vehicles - Diagnostic services - Part 1: Specification and requirements

[3] **ISO 15765-3**/Road vehicles - Diagnostics on controller area network (CAN) - Part 3: Implementation of diagnostic services

[4] **ISO/DIS 15765-2**/Road vehicles -- Diagnostic communication over Controller Area Network (DoCAN) -- Part 2: Transport protocol and network layer services

[5] http://www.repairfaq.org/filipg/LINK/F_crc_v34.html

[6] ASAM AE_MCD-1 XCP CAN-Transport-Layer Version 1.2.0

[7] ASAM AE_MCD-1 XCP Ethernet-Transport-Layer Version 1.2.0

[8] ASAM AE_MCD-1 XCP Flexray-Transport-Layer Version 1.2.0

[9] ASAM AE_MCD-1 XCP SxI-Transport-Layer Version 1.2.0

[10] ASAM AE_MCD-1 XCP USB-Transport-Layer Version 1.2.0

[11] ASAM AE Common SeedKey-and-checksum-Calculation Version 1.0.0

E-mail: info@asam.net
Internet: www.asam.net