Association for Standardisation of
Automation and Measuring Systems

ASAM MCD-1 (XCP on USB)

Universal Measurement and Calibration Protocol

USB Transport Layer

Version 1.5.0

Date: 2017-11-30

Associated Standard

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 separated 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

This associated standard describes the XCP on USB transport layer.

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

This standard describes how XCP is transported on USB as transport layer. It is shown how addressing shall be realized and the usage of the different communication models (see Chapter 4.3). Also the content of the control field of the XCP message frame format is described. For details about the frame format structure please refer the base standard [1]. Chapter 5 shows the specific commands which are defined for the transport on CAN. The interface to the ASAM MCD-2 MC description file is described in chapter 7.

3 RELATIONS TO OTHER STANDARDS

3.1 BACKWARD COMPATIBILITY TO EARLIER RELEASES

3.1.1 USB TRANSPORT LAYER

This Transport layer uses the version number 1.5. This version number is represented as 16 bit value, where the high byte contains the major version (U) and low byte contains the minor version (V) number.

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

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

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

3.1.2 THE COMPATIBILITY MATRIX

The Compatibility Matrix gives an overview of the allowed combinations of Protocol Layer and Transport Layer parts. For details about the Compatibility Matrix please refer the base standard [1].

3.2 REFERENCES TO OTHER STANDARDS

For details about the References to other standards please refer the base standard [1].

4 THE XCP TRANSPORT LAYER FOR USB

4.1 SETUP OF SLAVE BEFORE STARTING XCP COMMUNICATION

4.1.1 USB VERSION

The XCP on USB Transport Layer operates in minimum with USB Specification [3].

4.1.2 USB TRANSFER TYPES

XCP on USB allows only the transfer types bulk and interrupt.

4.1.3 USB CONFIGURATION, INTERFACE AND ALTERNATE SETTING

The host is responsible for configuring a USB device. As part of the configuration process, the host sets the device configuration and, where necessary, selects the appropriate alternate setting for the interface. These configurations must be set before XCP communication starts.
A configuration has one or more interfaces. There is only one configuration allowed with number one for a XCP slave device. The default configuration value is zero, when the device is attached to the host. Then the device supports only limited adjustments to the USB configuration and XCP communication is not available. The new configuration value one is set by the request `SetConfiguration()`. Then that configuration is selected and the device enters the Configured state. The host is not allowed to reset to configuration number zero because of unintended interruption of transmission and data lost.
A alternate setting can redefine the number or characteristics of the associated endpoints. If this is the case, the USB device must support the `GetInterface()` request to report the current alternate setting for the special interface and `SetInterface()` request to select the alternate setting. The slave device can execute `SetInterface()` only in Configured state.
The default setting when a device is initially configured is alternate setting zero. Other alternate settings are optional for a XCP slave device. When there is a alternate setting given by the slave device description file, then the host selects this alternate setting by `SetInterface()`. The host is not allowed to change the alternate setting any more after the first `SetInterface()`.
If there is no alternate setting in the slave device description file, the alternate setting zero remains active.

`SetConfiguration()`, `GetInterface()` and `SetInterface()` are standard device requests defined by USB specification.

4.2 ADDRESSING

USB devices report their attributes using descriptors. A descriptor is a data structure with a defined format. Among the standard descriptors the USB device descriptor contains helpful information for the identification of the attached USB device. It includes information that applies globally to the device and all of the device’s configurations. A USB device has

only one device descriptor. The host does an upload of this information during enumeration of the device.

4.2.1 IDENTIFICATION OF XCP SLAVE TO MASTER

The device descriptor provides a Vendor-ID and a Product-ID. Each parameter is coded by two bytes.
Apart from the device descriptor an interface descriptor describes information about the specific set of USB endpoints for XCP communication. The interface number must be known by the XCP master since there might be several interfaces on a single device, e.g. composite devices.
Vendor-ID, Product-ID and interface number are defined in the slave device description file (e.g. the MCD-2 MC format description file). The device indicates to the XCP master by Vendor-ID, Product-ID and interface number that there is an XCP slave attached.
The host configuration manager decides about the device driver to be loaded when a USB device is attached. The configuration manager selects the device driver depending on Vendor ID, Product-ID and interface number.

4.2.2 MULTIPLE SLAVE DEVICES OF THE SAME TYPE

XCP on USB allows communication of two or more slave devices to one host at the same time. When the slave devices differ in Vendor-ID and Product-ID there is the possibility of an unambiguous differentiation on USB level and the assignment of slave device to its description file is clear.
Slave devices of the same Vendor-ID and Product-ID require additional distinctive marks. Those slave devices must be clearly identified for binding by USB and assigning their slave description file. The USB serial number and the interface string descriptor accomplish these requirements.
USB serial number and interface string descriptor are only mandatory for slave devices of the same type, means same Vendor-ID and Product-ID.

4.2.2.1 USB SERIAL NUMBER

The device descriptor contains an index of the string descriptor describing the device's serial number. iSerialNumber refers to the string descriptor for the serial number.
If two or more slave devices of the same Vendor-ID and Product-ID can be attached to the host at the same time, each slave device has to provide a unique serial number.

4.2.2.2 USB INTERFACE STRING DESCRIPTOR

The field Interface contains an index to a string descriptor describing the interface. The content of a string descriptor is implementation specific but unique for slave devices of the same Vendor-ID and Product-ID. The XCP master compares the interface string with the ASCII string defined by the slave description file. The XCP master ignores the language information of the UNICODE interface string when comparing the strings.

4.2.3 RELATION OF XCP PACKETS TO DEVICE ENDPOINTS

A slave device connected by USB protocol is addressed by its device address and its device endpoint. Devices are assigned a unique device address by the USB System Software. A device endpoint is a uniquely addressable portion of a USB device that is the source or sink of information.

XCP on USB uses at least two different device endpoints: one for the CMD and STIM packets and one for the RES, ERR, EV, SERV and DAQ packets.
The assignment of device endpoints to the XCP objects CMD/STIM and RES/ERR/EV/SERV/DAQ is defined in the slave device description file (e.g. the MCD-2 MC format description file), which is used to configure the master device.

4.3 COMMUNICATION MODEL

XCP on USB makes use of the standard communication model.

The block transfer communication is optional.
The interleaved communication model is optional.

4.4 HEADER AND TAIL

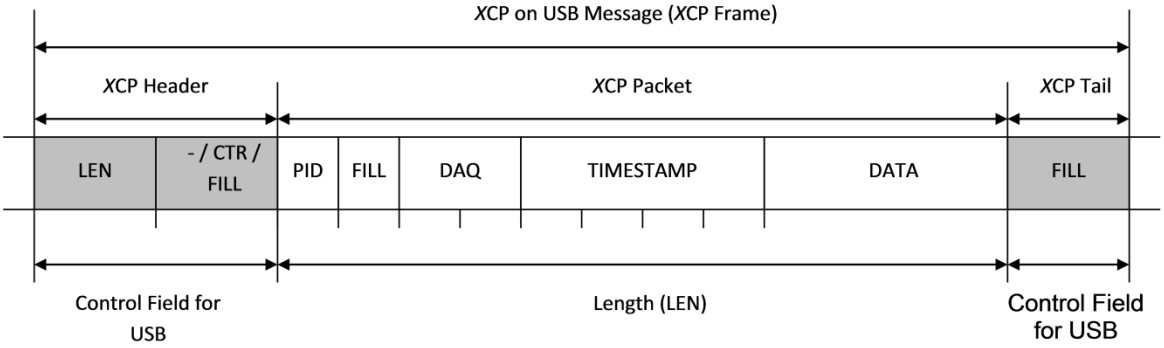

Figure 1 Header and tail for XCP on USB

4.4.1 HEADER

For XCP on USB the Header consists of a Control Field containing a **LEN**gth (LEN) and an optional **C**oun**T**e**R** (CTR) or an optional FILL.
The slave device description file (a2l file) informs the master about the Header Type.

4.4.1.1 LENGTH

LEN is the number of bytes in the original XCP Packet. LEN can be BYTE or WORD (Intel format). Please refer also to chapter 4.6 Packing of XCP Messages Into USB Data Packets.

4.4.1.2 COUNTER

The CTR value in the XCP Header allows detecting missing Packets.
The master has to generate a CTR value when sending a CMD or STIM message. The CTR value must be incremented for each new packet sent from master to the slave. In case of transmission of CMD and STIM messages on separate USB Endpoints, this requires separate CTR's per each USB Endpoint.
The slave has to generate a (independent) CTR value when sending RES/ERR, EV/SERV or DAQ messages. The CTR value must be incremented for each new packet sent from slave to the master. In case of transmission of RES/ERR, EV/SERV and DAQ messages on separate USB Endpoints, this requires separate CTR's per each USB Endpoint.
If available, CTR always has the same size as LEN.

4.4.1.3 FILL

The performance can be increased for accessing the XCP Messages' contents when a XCP Packet starts on an aligned address. The contents of the FILL bytes are uncertain.
If available, FILL always has the same size as LEN.

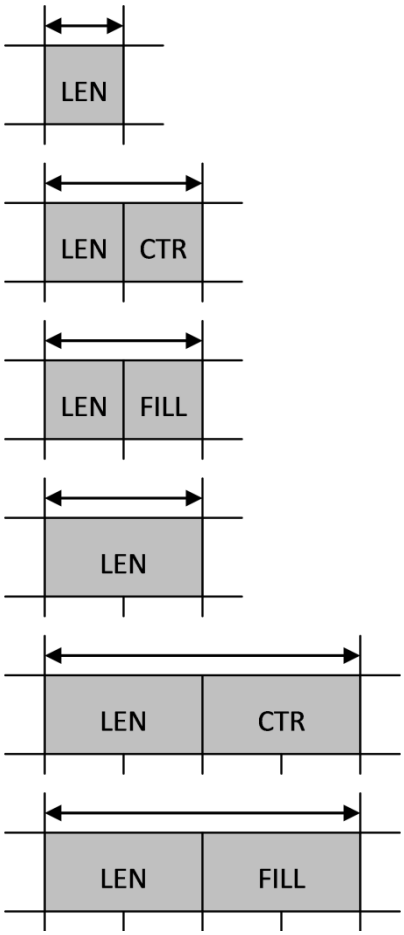

Figure 2 Header types for XCP on USB

4.4.2 TAIL

Depending on the packing configuration one or more XCP Messages reside within the USB data packet. The address must be properly aligned for the beginning of the next XCP Message. Therefore the length of the total XCP Message is always aligned to 8, 16, 32 or 64 bits. The slave device description file contains the value for alignment. It is 8 bit alignment by default. The contents of the FILL bytes are uncertain.

4.5 THE LIMITS OF PERFORMANCE

The ranges of MAX_CTO and MAX_DTO depend on the transfer mode used and the referring packing type.
The properties of transfer modes are summarized by the USB specification [3].

Table 1 CTO and DTO range

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

4.6 PACKING OF XCP MESSAGES INTO USB DATA PACKETS

To make optimal use of USB, multiple XCP Messages may be combined into a single USB data packet. XCP Messages can be also transmitted in streams crossing an USB data packet boundary. The packing of XCP Messages depends on slave device's capability.
There are three kinds for the packing:

- Single XCP Message in one USB data packet
- Multiple XCP Messages (a variable number of XCP Messages) within one USB data packet
- Streaming Mode

The slave description file (e.g. [2]) defines individually the kind of packing type for each device endpoint.
Multiple XCP Messages and Streaming Mode are optional.

4.6.1 SINGLE XCP MESSAGE IN ONE USB DATA PACKET

There is only one XCP Message in one USB data packet. The tail ensures that even after a shortened USB data packet the alignment of the next XCP Message is properly fulfilled.

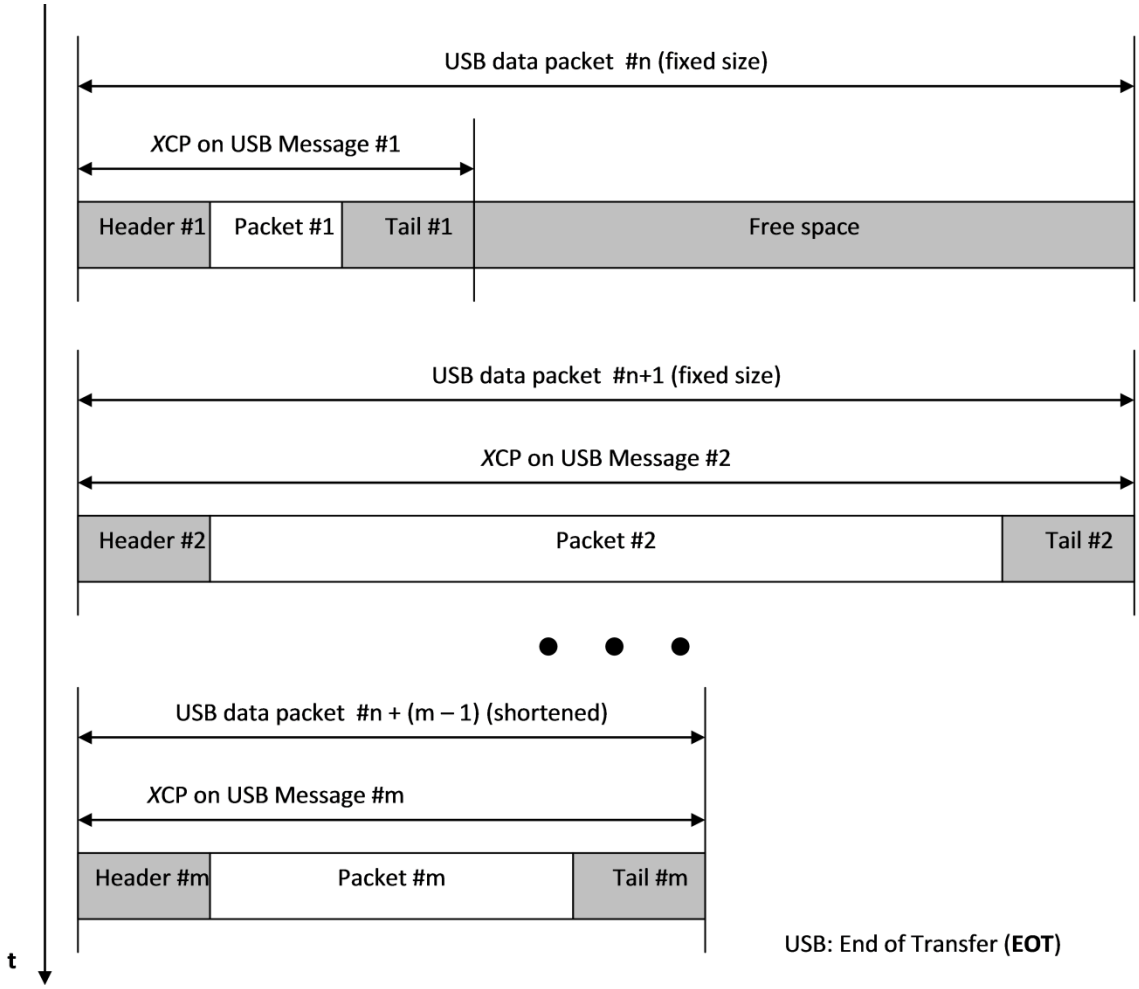

Figure 3 Example of single XCP message in one USB data packet

4.6.2 MULTIPLE XCP MESSAGES WITHIN ONE USB DATA PACKET

One or more XCP Messages can be combined but any XCP Message must not cross a USB packet boundary.
The XCP master or the XCP slave device concatenate XCP Messages only separated by tails accordingly to the required alignment. Please refer to chapter 4.4.2 Tail.
For details about holding on a transfer see also chapter 4.6.4 Filling up USB Data Packets.
The Figure 4 illustrates the options on putting multiple XCP Messages together in one USB data packet.

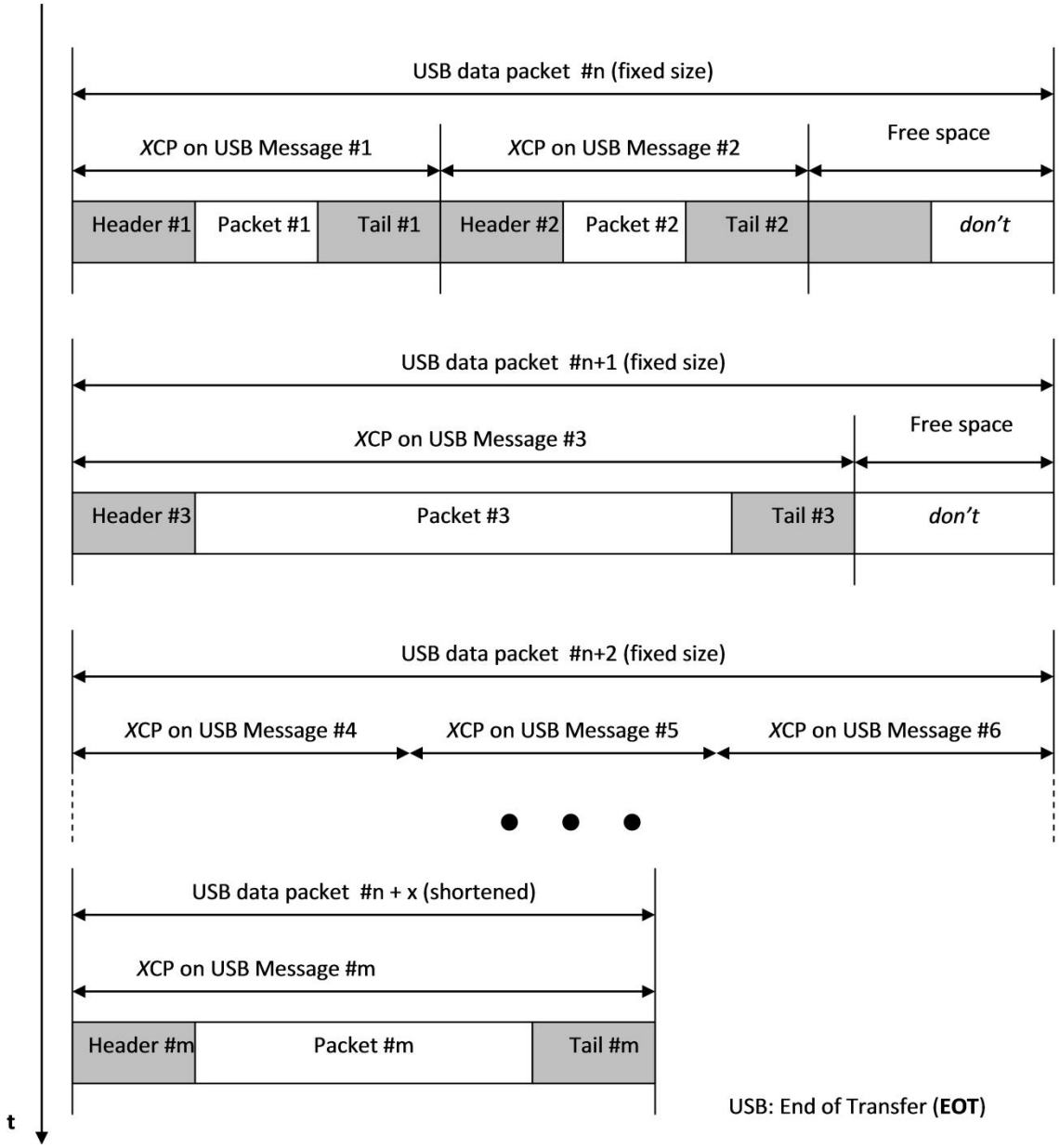

Figure 4 Example of multiple XCP messages within one USB data packet

4.6.3 STREAMING MODE

The XCP on USB Transport Layer allows putting XCP Messages together in a more fluently manner. A XCP Message may cross a USB packet boundary.
The XCP master or the XCP slave device concatenate XCP Messages only separated by tails accordingly to the required alignment. Please refer to chapter 4.4.2 Tail.
For details about holding on a transfer see also chapter 4.6.4 Filling up USB Data Packets.
The example below shows a transmission of three XCP Messages, e.g. three DAQ packets after data acquisition for a certain time. The USB transfer ends when the last XCP Message is sent.

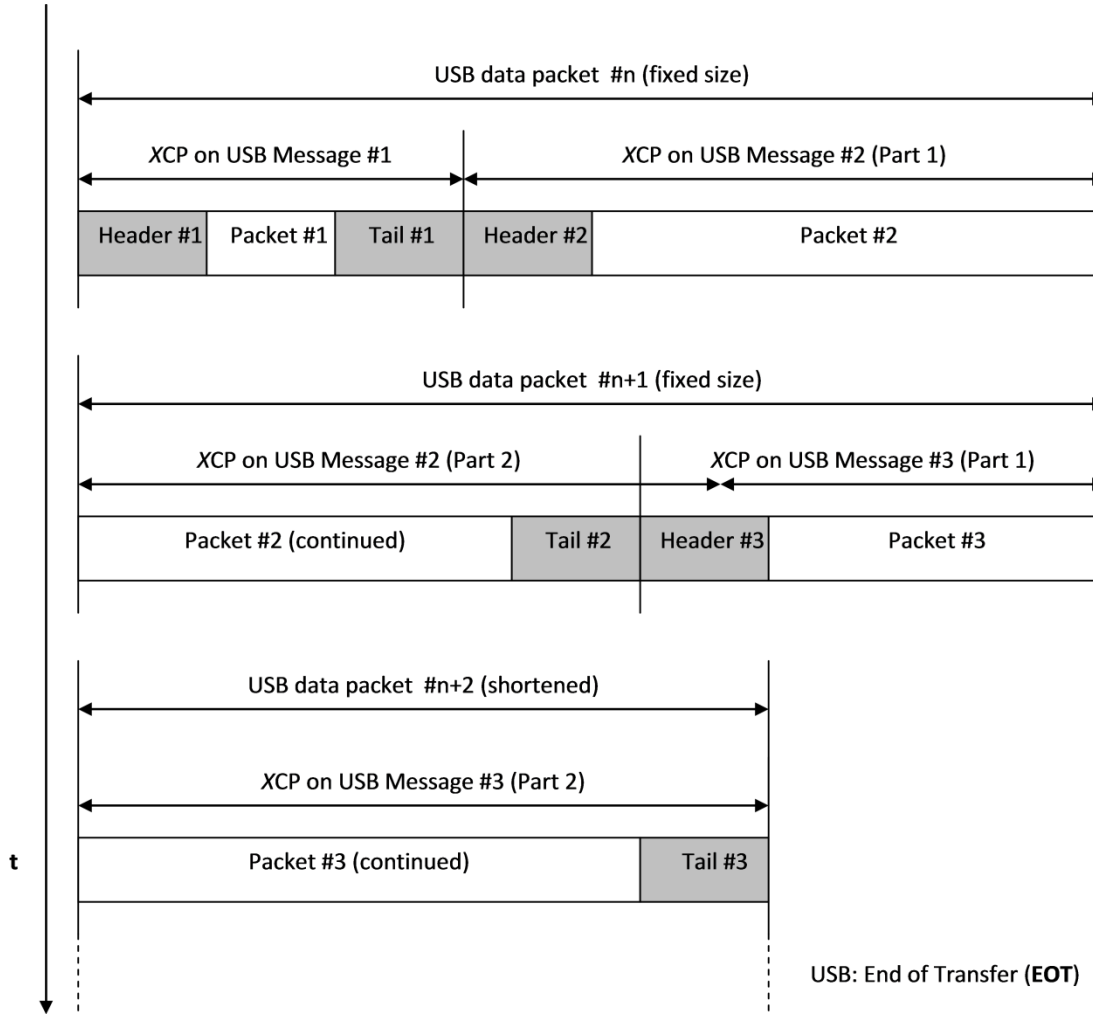

Figure 5 Example of streaming mode with EOT

4.6.4 FILLING UP USB DATA PACKETS

Communication between XCP master and slave takes place by one or more USB transfers. A USB transfer consists of one or more USB data packets. An USB data packet, smaller than the maximum packet size, aborts the current transfer. For all packing types the USB data packet can be filled up with dummy data preventing from abortion of transfer. It is implementation specific filling up USB data packets and not mandatory in terms of XCP communication. The contents of fill bytes do not care.

The packing types Multiple XCP Messages or Streaming Mode require a mechanism to indicate that there is no more XCP Message in the current USB data packet.
In both packing types the receiver will stop extracting of further XCP Messages from a USB data packet when one of the following conditions is met:

- The length (LEN) equals zero.
- The remaining space in the USB data packet is less than the size of the XCP Header.
- There is no more data because it is a shortened USB data packet indicating the end of transfer (EOT).

In case of a message length of zero, the receiver neglects the XCP Message contents and the counter CTR stands still.
A new XCP Message ready for transmission starts at the beginning of the next USB data packet.

5 SPECIFIC COMMANDS FOR XCP ON USB

Table 2 Command codes overview

Command	Code	Timeout	Remark
GET_DAQ_EP	0xFF	t_1	Optional
SET_DAQ_EP	0xFE	t_1	Optional

5.1 GET DAQ LIST USB ENDPOINT

Category USB only, optional
Mnemonic GET_DAQ_EP

Table 3 GET_DAQ_EP command structure

Position	Type	Description
0	BYTE	Command Code = TRANSPORT_LAYER_CMD = 0xF2
1	BYTE	Sub Command Code = GET_DAQ_EP = 0xFF
2, 3	WORD	DAQ_LIST_NUMBER [0,1,...MAX_DAQ-1]

Positive Response:

Table 4 GET_DAQ_EP positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	USB_ENDPOINT_FIXED
		0 = USB Endpoint can be configured
		1 = USB Endpoint is fixed
2, 3	WORD	Reserved
4	BYTE	USB Endpoint Number of DTO dedicated to list number

As a default, the master transfers all DAQ lists with DIRECTION = STIM on the same USB Endpoint as used for CMD.
Alternatively, the master may have individual USB Endpoints (other than the one used for CMD) for the DAQ lists with DIRECTION = STIM.
As a default, the slave transfers all DAQ lists with DIRECTION = DAQ on the same USB Endpoint as used for RES/ERR and EV/SERV.
Alternatively, the slave may have individual USB Endpoints (other than the one used for RES/ERR and EV/SERV) for its DAQ lists with DIRECTION = DAQ.
With GET_DAQ_EP, the master can detect whether a DAQ list uses an individual USB Endpoint and whether this Endpoint is fixed or configurable.

If the USB Endpoint is configurable, the master can configure the individual USB Endpoint for this DAQ list with `SET_DAQ_EP`.

5.2 SET DAQ LIST USB ENDPOINT

Category USB only, optional
Mnemonic SET_DAQ_EP

Table 5 SET_DAQ_EP command structure

Position	Type	Description
0	BYTE	Command Code = TRANSPORT_LAYER_CMD = 0xF2
1	BYTE	Sub Command Code = SET_DAQ_EP = 0xFE
2, 3	WORD	DAQ_LIST_NUMBER [0,1,...MAX_DAQ-1]
4	BYTE	USB Endpoint of DTO dedicated to list number

The master can assign an individual USB Endpoint to a DAQ list.
If the given USB Endpoint is not possible, the slave returns an ERR_OUT_OF_RANGE.

5.3 COMMUNICATION ERROR HANDLING

This chapter describes transport layer specific error handling. It extends the error handling concepts specified in the chapter “Communication Error Handling” of the base standard [1]. Please refer to the base standard for obtaining fundamentals on XCP error handling.

5.3.1 ERROR CODE HANDLING

Table 6 Transport Layer Ethernet subcommands error handling

Command	Error	Pre-Action	Action
GET_DAQ_EP	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_SUBCMD_UNKNOWN	-	display error or silently ignore error
SET_DAQ_EP	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_SUBCMD_UNKNOWN	-	display error or silently ignore error

6 SPECIFIC EVENTS FOR XCP ON USB

There are no specific events for XCP on USB at the moment.

7 INTERFACE TO ASAM MCD-2 MC DESCRIPTION FILE

The following chapter describes the parameters that are specific for XCP on USB.

7.1 ASAM MCD-2 MC AML FOR XCP ON FLEXRAY

The AML for the XCP on USB transport layer specific properties is defined in the file named XCP_vX_Y_on_USB.aml where vX_Y is the current protocol layer version.

7.2 IF_DATA EXAMPLE FOR XCP ON USB

The file XCP_vX_Y_IF_DATA_example.a2l where vX_Y is the current protocol layer version gives an IF_DATA example for a XCP on USB transport layer (see section beginning with “/begin XCP_ON_USB”).

8 SYMBOL AND ABBREVIATED TERMS

A2L	**A**SAM MCD-**2** MC **L**anguage File
ASCII	**A**merican **S**tandard for **C**haracter **I**nformation **I**nterchange
CAN	**C**ontroller **A**rea **N**etwork
CCP	**C**AN **C**alibration **P**rotocol
CMD	**Co**m**m**an**d**
CTO	**C**ommand **T**ransfer **o**bject
CTR	**C**oun**te**r
DAQ	**D**ata **Ac**quisition
DTO	**D**ata **T**ransfer **O**bjects
ECU	**E**lectronic **C**ontrol **U**nit
EOT	**E**nd **o**f **T**ransfer
ERR	**Err**or
EV	**Ev**ent
ID	**Id**entifier
IP	**I**nternet **P**rotocol
LEN	**Len**gth
PID	**P**acket **Id**entifier
RES	**Res**ponses
SCI	**S**erial **C**ommunication **I**nterface
SERV	**Serv**ice
SPI	**S**erial **P**eripheral **I**nterface
STIM	Synchronous Data **Stim**ulation
SxI	**S**erial **X** **I**nterface
TCP	**T**ransfer **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
USB	**U**niversal **S**erial **B**us
XCP	Universal Measurement and **C**alibration **P**rotocol

9 BIBLIOGRAPHY

[1] ASAM AE MCD-1 XCP BS Protocol-Layer BS Version V1.3.0
[2] **ASAM MCD-2 MC**/"Measurement and Calibration Data Specification", Version 1.7.x
[3] Universal Serial Bus Specification Version 1.1, 2.0 and 3.0

Figure Directory

Table Directory

Association for Standardisation of
Automation and Measuring Systems

E-mail: support@asam.net
Web: www.asam.net