XCP
Version 1.1

Part 3 - Transport Layer Specification XCP on USB

Part 3 – XCP on USB – Transport Layer Specification

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

Dated:31-03-2008

Status of Document

Date:	31-03-2008
Author:	Roel Schuermans, Vector Informatik GmbH Andreas Zeiser, Vector Informatik GmbH Oliver Kitt, Vector Informatik GmbH Hans-Georg Kunz, VDO Automotive AG Hendirk Amsbeck, dSPACE GmbH Bastian Kellers, dSPACE GmbH Boris Ruoff, ETAS GmbH Reiner Motz, Robert Bosch GmbH Dirk Forwick, Robert Bosch GmbH
Version:	Version 1.1
Doc-ID:	
Status:	Release
Type	

Revision History

This revision history shows only major modifications between release versions.

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

Table of contents

Table of diagrams:

0 INTRODUCTION

0.1 THE *X*CP PROTOCOL FAMILY

This document is based on experiences with the **C**AN **C**alibration **P**rotocol (CCP) version 2.1 as described in feedback from the companies Accurate Technologies Inc., Compact Dynamics GmbH, DaimlerChrysler AG, dSPACE GmbH, ETAS GmbH, Kleinknecht Automotive GmbH, Robert Bosch GmbH, Siemens VDO Automotive AG and Vector Informatik GmbH.
The *X*CP Specification documents describe an improved and generalized version of CCP.
The generalized protocol definition serves as standard for a protocol family and is called "*X*CP" (Universal Measurement and **C**alibration **P**rotocol).
The **"*X*"** generalizes the "various" transportation layers that are used by the members of the protocol family e.g "*X*CP on CAN", "*X*CP on TCP/IP", "*X*CP on UDP/IP", "*X*CP on USB" and so on.

*X*CP is not backwards compatible to an existing CCP implementation.

0.2 DOCUMENTATION OVERVIEW

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

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

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

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

This document describes the way how the *X*CP protocol is transported on USB.

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

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

Related Standards:
- ASAP2 V1.4 (at least)
- XCP V1.0
- USB1.1, USB2.0

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

0.3 DEFINITIONS AND ABBREVIATIONS

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

Abbreviation	Description
A2L	File Extension for an **A**SAM **2**MC **L**anguage File
AML	**A**SAM 2 **M**eta **L**anguage
ASAM	**A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems
BYP	**BYP**assing
CAL	**CAL**ibration
CAN	**C**ontroller **A**rea **N**etwork
CCP	**C**an **C**alibration **P**rotocol
CMD	**C**o**M**man**D**
CS	**C**heck**S**um
CTO	**C**ommand **T**ransfer **O**bject
CTR	**C**oun**T**e**R**
DAQ	**D**ata **A**c**Q**uisition, **D**ata **A**c**Q**uisition Packet
DTO	**D**ata **T**ransfer **O**bject
ECU	**E**lectronic **C**ontrol **U**nit
ERR	**ERR**or Packet
EV	**EV**ent Packet
Host	The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operation system in use. (Definition copied from USB specification 2.0)
LEN	**LEN**gth
MCD	**M**easurement **C**alibration and **D**iagnostics
MTA	**M**emory **T**ransfer **A**ddress
ODT	**O**bject **D**escriptor **T**able
PAG	**PAG**ing
PGM	**P**ro**G**ra**M**ming
PID	**P**acket **ID**entifier
RES	command **RES**ponse packet
SERV	**SERV**ice request packet
STD	**ST**an**D**ard
STIM	Data **STIM**ulation packet
TCP/IP	**T**ransfer **C**ontrol **P**rotocol **/ I**nternet **P**rotocol
TS	**T**ime Stamp
UDP/IP	**U**nified **D**ata **P**rotocol **/ I**nternet **P**rotocol
USB	**U**niversal **S**erial **B**us
USB data packet	A USB data packet is one of the three types of packets forming the communication between host and device. It carries the original information (pay load) as part of all or in total.
USB interface	A unique set of USB endpoints forms the USB interface. A composite device contains at least two different interfaces applying to different software clients.
XCP	Universal **C**alibration **P**rotocol
*X*CP Frame	Same as *X*CP Message. *X*CP Message is used consequently in the following description.

Table 1: Definitions and Abbreviations

0.4 MAPPING BETWEEN XCP DATA TYPES AND ASAM DATA TYPES

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

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

1 THE *X*CP TRANSPORT LAYER FOR USB

1.1 SETUP OF SLAVE BEFORE STARTING XCP COMMUNICATION

1.1.1 USB VERSION

The *X*CP on USB Transport Layer operates in minimum with USB Specification Revision 1.1 and USB Specification Revision 2.0.

1.1.2 USB TRANSFER TYPES

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

1.1.3 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 *X*CP communication starts.

A configuration has one or more interfaces. There is only one configuration allowed with number one for a *X*CP 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 *X*CP 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 *X*CP 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.

1.2 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.

1.2.1 IDENTIFICATION OF *X*CP 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 *X*CP communication. The interface number must be known by the *X*CP 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 ASAP2 format description file). The device indicates to the *X*CP master by Vendor-ID, Product-ID and interface number that there is an *X*CP 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.

1.2.2 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 a 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.

1.2.2.1 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.

1.2.2.2 USB INTERFACE STRING DESCRIPTOR

The field *iInterface* 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 *X*CP 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.

1.2.3 RELATION OF *X*CP 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.

*X*CP 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 *X*CP objects CMD/STIM and RES/ERR/EV/SERV/DAQ is defined in the slave device description file (e.g. the ASAP2 format description file), which is used to configure the master device.

1.3 COMMUNICATION MODEL

*X*CP on USB makes use of the standard communication model.

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

1.4 HEADER AND TAIL

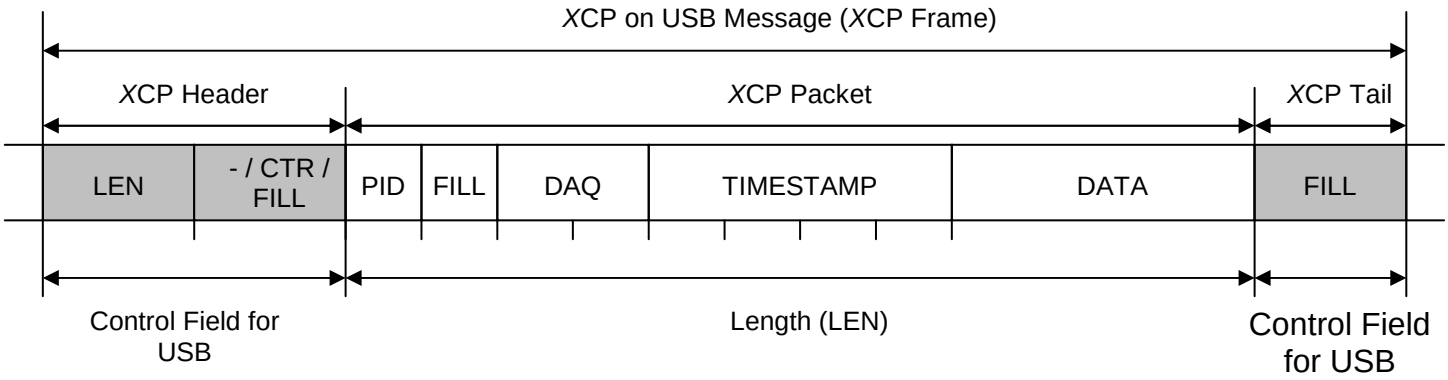

Diagram 1 : Header and Tail for *X*CP on USB

1.4.1 HEADER

For *X*CP 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.

1.4.1.1 LENGTH

LEN is the number of bytes in the original *X*CP Packet. LEN can be BYTE or WORD (Intel format). Please refer also to chapter 1.6 Packing of *X*CP Messages into USB data packets.

1.4.1.2 COUNTER

The CTR value in the *X*CP Header allows to detect 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.

1.4.1.3 FILL

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

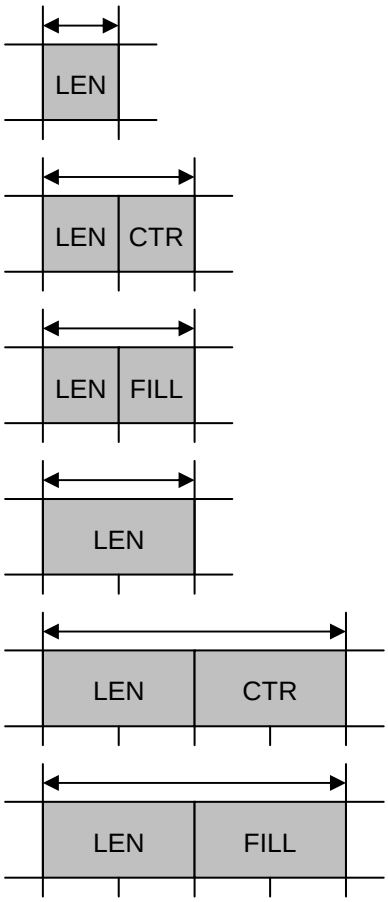

Diagram 2 : Header Types for *X*CP on USB

1.4.2 TAIL

Depending on the packing configuration one or more *X*CP Messages reside within the USB data packet. The address must be properly aligned for the beginning of the next *X*CP Message. Therefore the length of the total *X*CP 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.

1.5 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 (e.g. revision 1.1 / 2.0)

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

1.6 PACKING OF *X*CP MESSAGES INTO USB DATA PACKETS

To make optimal use of USB, multiple *X*CP Messages may be combined into a single USB data packet. *X*CP Messages can be also transmitted in streams crossing an USB data packet boundary. The packing of *X*CP Messages depends on slave device's capability.

There are three kinds for the packing:

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

The slave description file (e.g. the ASAP2 format description file) defines individually the kind of packing type for each device endpoint.

Multiple XCP Messages and Streaming Mode are optional.

1.6.1 SINGLE *X*CP MESSAGE IN ONE USB DATA PACKET

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

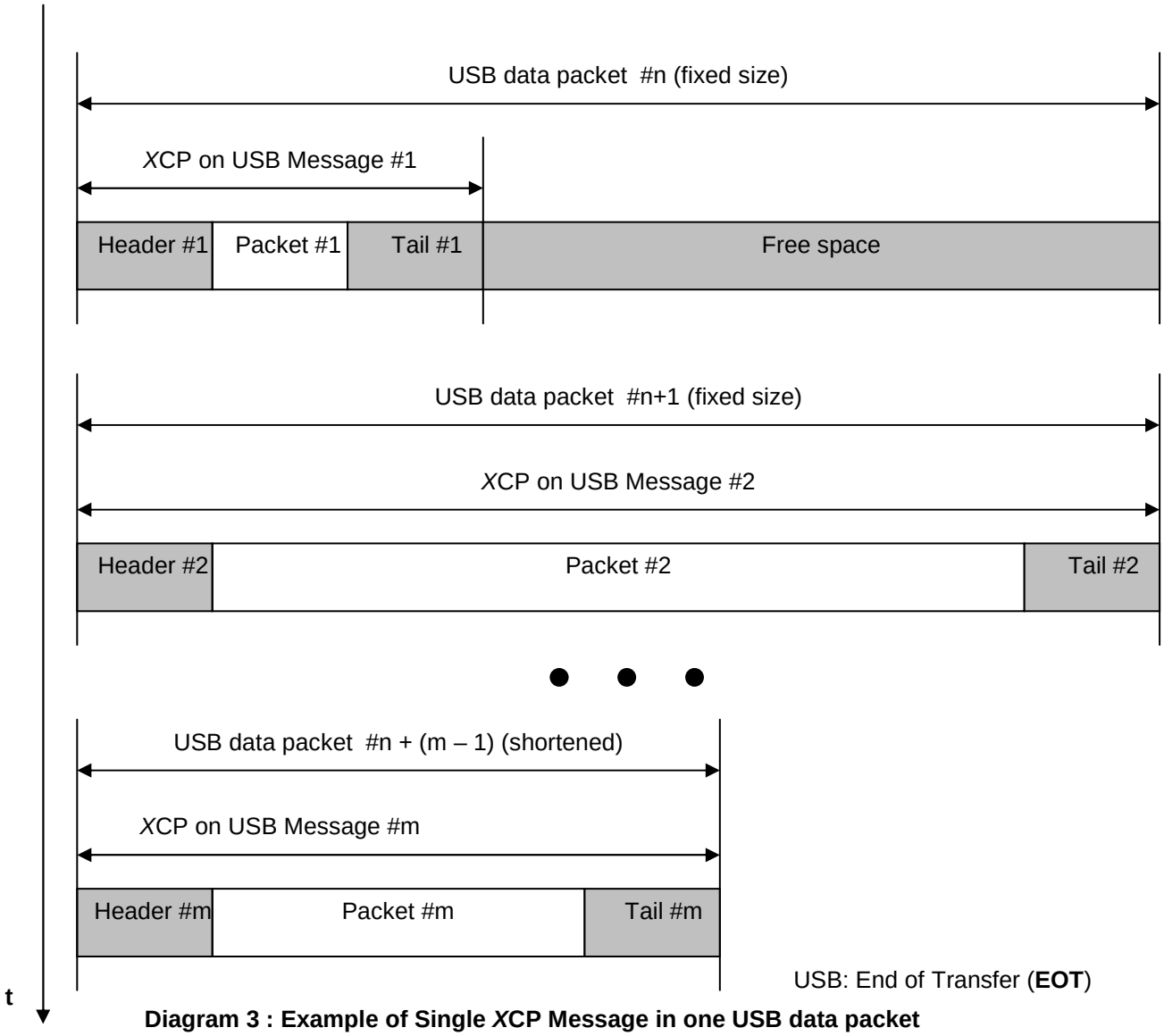

Diagram 3 : Example of Single *X*CP Message in one USB data packet

1.6.2 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 1.4.2 Tail.

For details about holding on a transfer see also chapter 1.6.4 Filling up USB data packets.

The Diagram 4 illustrates the options on putting multiple *X*CP Messages together in one USB data packet.

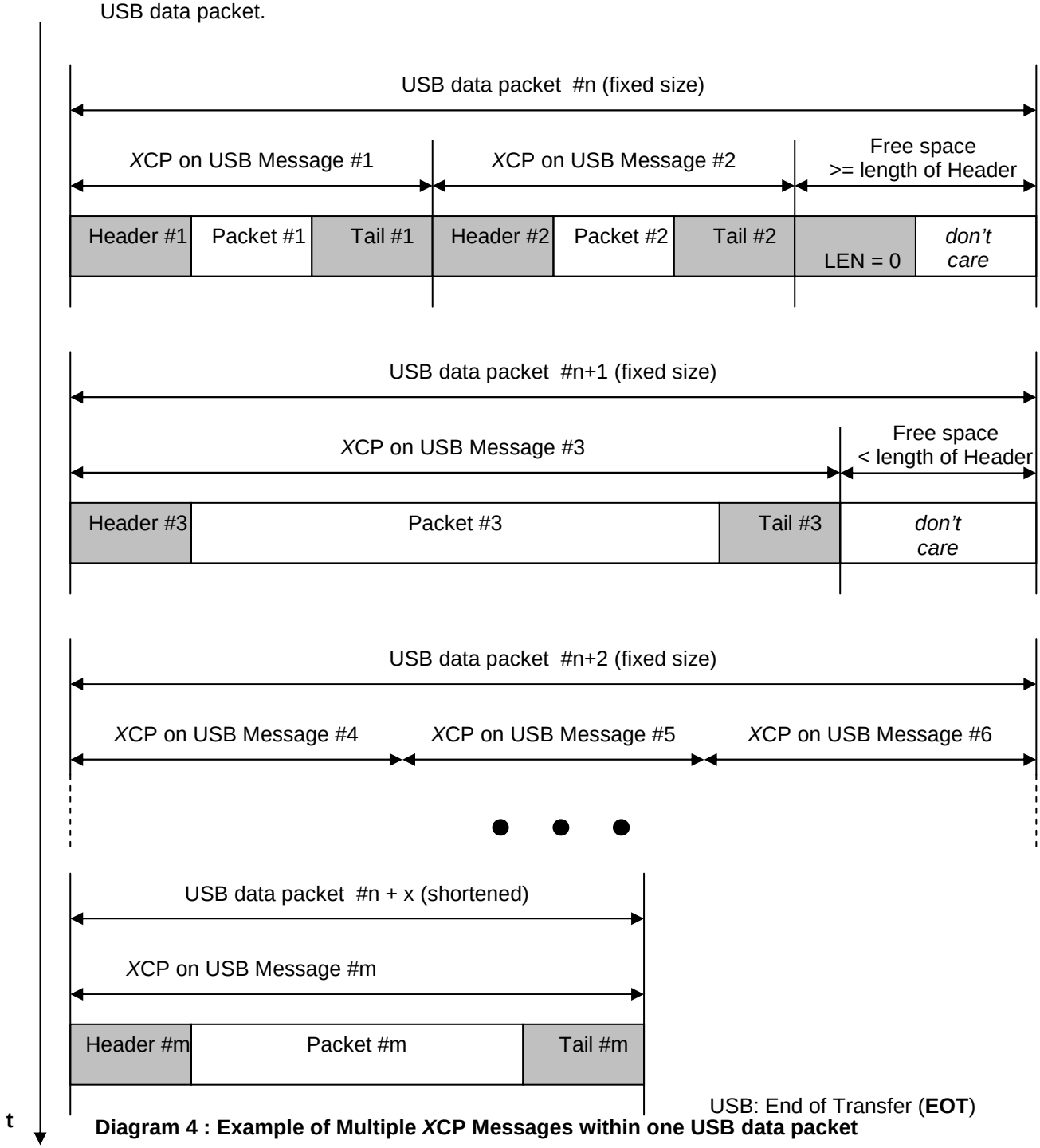

Diagram 4 : Example of Multiple *X*CP Messages within one USB data packet

1.6.3 STREAMING MODE

The *X*CP on USB Transport Layer allows putting *X*CP Messages together in a more fluently manner. A *X*CP Message may cross a USB packet boundary.

The *X*CP master or the *X*CP slave device concatenate *X*CP Messages only separated by tails accordingly to the required alignment. Please refer to chapter 1.4.2 Tail.

For details about holding on a transfer see also chapter 1.6.4 Filling up USB data packets.

The example below shows a transmission of three *X*CP Messages, e.g. three DAQ packets after data acquisition for a certain time. The USB transfer ends when the last *X*CP Message is sent.

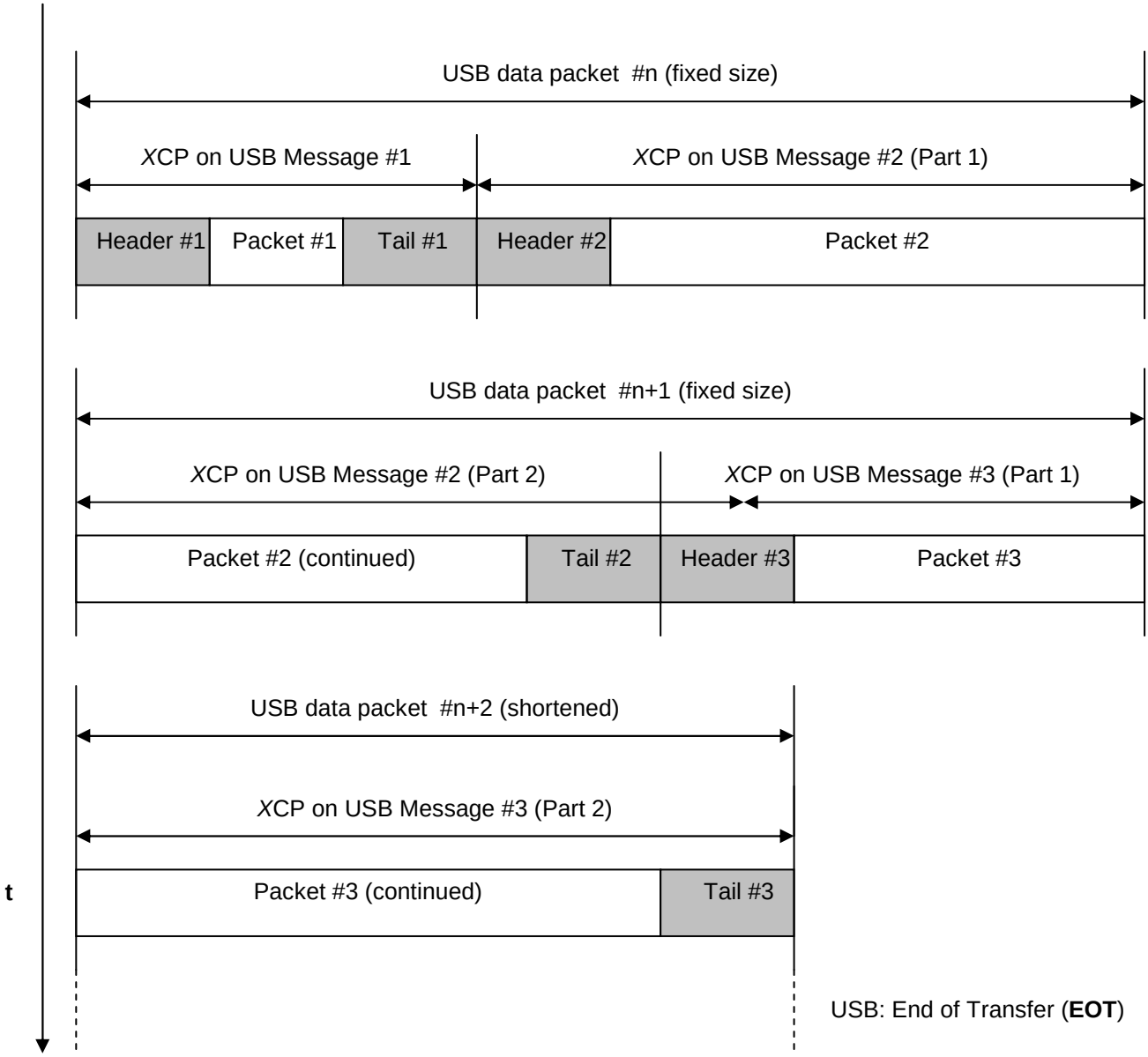

Diagram 5 : Example of Streaming Mode with EOT

1.6.4 FILLING UP USB DATA PACKETS

Communication between *X*CP master and slave takes place by one or more USB transfers. A USB transfer consists of one or more USB data packets. A 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 *X*CP communication. The contents of fill bytes don't care.

The packing types Multiple *X*CP Messages or Streaming Mode require a mechanism to indicate that there is no more *X*CP Message in the current USB data packet.

In both packing types the receiver will stop extracting of further *X*CP 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 *X*CP Header.
- There is no more data because it is an shortened USB data packet indicating the end of transfer (EOT).

In case of a message length of zero, the receiver neglects the *X*CP Message contents and the counter CTR stands still.

A new *X*CP Message ready for transmission starts at the beginning of the next USB data packet.

2 SPECIFIC COMMANDS FOR *XCP* ON USB

Table of Command Codes:

Command	Code	Timeout	Remark
GET_DAQ_EP	0xFF	t1	Optional
SET_DAQ_EP	0xFE	t1	Optional

2.1 GET DAQ LIST USB ENDPOINT

Category USB only, optional
Mnemonic GET_DAQ_EP

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:

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.

2.2 SET DAQ LIST USB ENDPOINT

Category USB only, optional
Mnemonic SET_DAQ_EP

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 isn't possible, the slave returns an ERR_OUT_OF_RANGE.

3 SPECIFIC EVENTS FOR *X*CP ON USB

There are no specific events for *X*CP on USB at the moment.

4 INTERFACE TO ASAM MCD 2MC DESCRIPTION FILE

The following chapter describes the parameters that are specific for *X*CP on USB.

4.1 ASAM MCD 2MC AML FOR *X*CP ON USB

```
/*****************************************************************************/
/*                                                                           */
/*   ASAP2 meta language for XCP on USB V1.0                                 */
/*   Assumes ASAP2 V1.4 or later                                             */
/*                                                                           */
/*   2003-12-16                                                              */
/*                                                                           */
/*   XCP on USB working group                                                */
/*                                                                           */
/*   Datatypes:                                                              */
/*                                                                           */
/* A2ML   ASAP2                  Windows         description                 */
/* ------------------------------------------------------------------------- */
/* uchar  UBYTE          BYTE            unsigned 8 Bit                      */
/* char   SBYTE          char            signed 8 Bit                        */
/* uint   UWORD          WORD            unsigned integer 16 Bit             */
/* int    SWORD          int             signed integer 16 Bit               */
/* ulong  ULONG          DWORD           unsigned integer 32 Bit             */
/* long   SLONG          LONG            signed integer 32 Bit               */
/* float  FLOAT32_IEEE                   float 32 Bit                        */
/*****************************************************************************/
/************************ start of USB ***************************************/
struct ep_parameters {
   uchar;                    /* ENDPOINT_NUMBER, not endpoint address */
   enum {
      “BULK_TRANSFER”        = 2,     /* Numbers according to USB spec.  */
      “INTERRUPT_TRANSFER”    = 3
   };
   uint;                     /* wMaxPacketSize: Maximum packet  */
                             /* size of endpoint in bytes       */
   uchar;                    /* bInterval: polling of endpoint  */

   enum {                          /* Packing of XCP Messages        */
      "MESSAGE_PACKING_SINGLE"    = 0, /* Single per USB data packet      */
      "MESSAGE_PACKING_MULTIPLE"  = 1, /* Multiple per USB data packet    */
      "MESSAGE_PACKING_STREAMING" = 2  /* No restriction by packet sizes  */
   };
```

```
        enum {                                  /* Alignment mandatory for all     */
          "ALIGNMENT_8_BIT" = 0,                /* packing types                   */
          "ALIGNMENT_16_BIT"= 1,
          "ALIGNMENT_32_BIT"= 2,
          "ALIGNMENT_64_BIT"= 3
        };

        taggedstruct {                  /* Optional */
          "RECOMMENDED_HOST_BUFSIZE" uint; /* Recommended size for the host   */
                                  /* buffer size. The size is defined*/
                                  /* as multiple of wMaxPacketSize.  */
        };
      }; /* end of ep_parameters */


      struct USB_Parameters {
        uint;                     /* XCP on USB version  */
                                  /* e.g. „1.0“ = 0x0100 */

        uint;                     /* Vendor ID                  */
        uint;                     /* Product ID                 */
        uchar;                      /* Number of interface          */

        enum {
          "HEADER_LEN_BYTE"      = 0,
          "HEADER_LEN_CTR_BYTE"  = 1,
          "HEADER_LEN_FILL_BYTE" = 2,
          "HEADER_LEN_WORD"      = 3,
          "HEADER_LEN_CTR_WORD"  = 4,
          "HEADER_LEN_FILL_WORD" = 5
        };

        /* OUT-EP for CMD and                                       */
        /* STIM (additional USB Endpoints may also be specified)    */
    taggedunion {
          block "OUT_EP_CMD_STIM" struct ep_parameters;
        };

        /* IN-EP for RES/ERR,                                       */
        /* DAQ (additional USB Endpoints may also be specified)     */
        /* and EV/SERV (if not specified otherwise)                 */
        taggedunion {
          block "IN_EP_RESERR_DAQ_EVSERV" struct ep_parameters;
        };
```

```
                        /* ---------- Begin of optional ------- */

    taggedstruct {                  /* Optional */
      "ALTERNATE_SETTING_NO" uchar;    /* Number of alternate setting     */

                        /* String Descriptor of XCP        */
                        /* interface                          * /
      "INTERFACE_STRING_DESCRIPTOR" char [101];
                        /* multiple OUT-EP's for STIM      */
      (block "OUT_EP_ONLY_STIM" struct ep_parameters)*;
                        /* multiple IN-EP's for DAQ        */
      (block "IN_EP_ONLY_DAQ" struct ep_parameters)*;
                        /* only one IN-EP for EV/SERV      */
      block "IN_EP_ONLY_EVSERV" struct ep_parameters;

      /* Not associated DAQ-Lists are assigned per default to           */
      /* OUT_EP_CD_STIM / IN_EP_RESERR_DAQ_EVSERV                       */
      (block "DAQ_LIST_USB_ENDPOINT" struct {
        uint;               /* reference to DAQ_LIST_NUMBER          */
        taggedstruct {        /* only mentioned if not VARIABLE       */
          "FIXED_IN" uchar;  /* this DAQ list always                 */
                        /* ENDPOINT_NUMBER, not endpoint address */
          "FIXED_OUT" uchar; /* this STIM list always                */
                        /* ENDPOINT_NUMBER, not endpoint address */
        };
      })*; /* end of DAQ_LIST_USB_ENDPOINT */
    }; /* end of optional */

};/************************* end of USB ***********************************/
```

4.2 IF_DATA EXAMPLE FOR *XCP* ON USB

```
/begin XCP_ON_USB

  0x0100                    /* XCP on USB version  */

  0x108C                    /* Vendor ID is 0x108C          */
  0x0EDC                    /* Product ID is 0x0EDC         */
  0x02                      /* Number of interface is 2     */

  HEADER_LEN_FILL_WORD

  /* OUT-EP for CMD and                                      */
  /* STIM (additional USB Endpoints may also be specified)   */
  /begin OUT_EP_CMD_STIM
    0x01                    /* Endpoint number is 1 (dir=OUT)  */
    BULK_TRANSFER
    0x40                    /* Maximum packet size is 64 bytes */
    0                       /* Polling interval don't care     */
    MESSAGE_PACKING_SINGLE  /* Only one XCP Message/packet     */
    ALIGNMENT_32_BIT        /* XCP Message starts on 32 bit    */
                            /* addresses within USB data packet*/
  /end OUT_EP_CMD_STIM

  /* IN-EP for RES/ERR,                                      */
  /* DAQ (additional USB Endpoints may also be specified)    */
  /* and EV/SERV (if not specified otherwise)                */
  /begin IN_EP_RESERR_DAQ_EVSERV
    0x01                    /* Endpoint number is 1 (dir=IN)   */
    BULK_TRANSFER
    0x40                    /* Maximum packet size is 64 bytes */
    0                       /* Polling interval don't care  */
    MESSAGE_PACKING_SINGLE /* Only one XCP Message/packet     */
    ALIGNMENT_32_BIT        /* XCP Message starts on 32 bit    */
                            /* addresses within USB data packet*/
    RECOMMENDED_HOST_BUFSIZE 1      /* Host: 1 * 64 bytes = 64 bytes   */
  /end IN_EP_RESERR_DAQ_EVSERV
```

```
                    /* ---------- Begin of optional ------- */

ALTERNATE_SETTING_NO 0x01          /* Use alternate setting number 1  */
                                   /* of XCP interface                */
                                   /* Identification for assignment   */
                                   /* description file                */
INTERFACE_STRING_DESCRIPTOR "XCP Master ECU on interface 1"

/begin OUT_EP_ONLY_STIM       /* Endpoint supporting packet type      */
                              /* only STIM                            */
   0x02                    /* Endpoint number is 2 (dir=OUT)  */
   BULK_TRANSFER
   0x40                    /* Maximum packet size is 64 bytes */
   0                       /* Polling interval don't care     */
   MESSAGE_PACKING_MULTIPLE   /* Multiple XCP Messages/packet    */
   ALIGNMENT_32_BIT           /* XCP Messages start on 32 bit    */
                              /* addresses within USB data packet*/
/end OUT_EP_ONLY_STIM

/begin IN_EP_ONLY_DAQ         /* Endpoint supporting packet type      */
                              /* only DAQ                             */
   0x02                       /* Endpoint number is 2 (dir=IN)   */
   BULK_TRANSFER
   0x40                    /* Maximum packet size is 64 bytes */
   0                       /* Polling interval don't care  */
   MESSAGE_PACKING_STREAMING   /* Streaming mode                  */
   ALIGNMENT_32_BIT            /* XCP Messages start on 32 bit    */
                               /* addresses within USB data packet*/
   RECOMMENDED_HOST_BUFSIZE 5   /* Host: 5 * 64 bytes = 320 bytes  */
/end IN_EP_ONLY_DAQ

/begin IN_EP_ONLY_EVSERV      /* Endpoint supporting packet types     */
                              /* only EV, SERV                        */
   0x03                       /* Endpoint number is 3 (dir=IN)   */
   INTERRUPT_TRANSFER
   0x40                    /* Maximum packet size is 64 bytes */
   0x10                    /* Polling interval is 16 ms       */
   MESSAGE_PACKING_SINGLE   /* Only one XCP Message/packet     */
   ALIGNMENT_32_BIT            /* XCP Message starts on 32 bit    */
                               /* addresses within USB data packet*/
   RECOMMENDED_HOST_BUFSIZE 1      /* Host: 1 * 64 bytes = 64 bytes   */
/end IN_EP_ONLY_EVSERV
```

```
    /begin DAQ_LIST_USB_ENDPOINT
      0x0000                  /* DTO-DAQ dedicated to list 0 */
      FIXED_IN 0x02             /* uses Endpoint 2 IN. */
    /end DAQ_LIST_USB_ENDPOINT

    /begin DAQ_LIST_USB_ENDPOINT
      0x0001                  /* DTO-DAQ dedicated to list 1 */
      FIXED_IN 0x02             /* uses Endpoint 2 IN. */
    /end DAQ_LIST_USB_ENDPOINT

    /begin DAQ_LIST_USB_ENDPOINT
      0x0002                  /* DTO-STIM dedicated to list 2 */
      FIXED_OUT 0x02            /* uses Endpoint 2 OUT. */
    /end DAQ_LIST_USB_ENDPOINT

  /end XCP_ON_USB
```

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

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