XCP
Version 1.1

Part 3 - XCP on CAN - Transport Layer Specification

Part 3 – XCP on CAN – 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

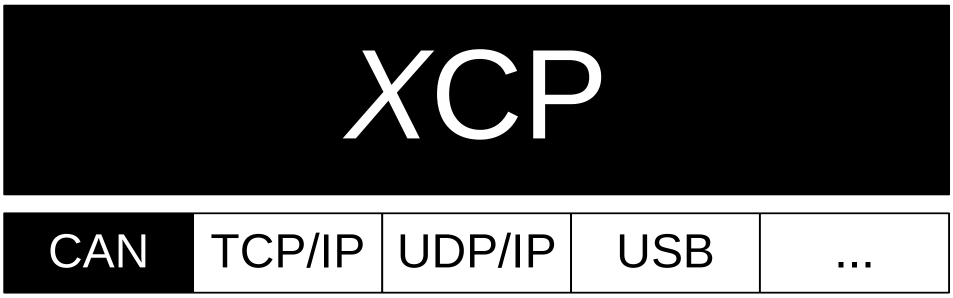
Table of contents

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 CAN.

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

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

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

0.3 Definitions and Abbreviations

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

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

Table 1: Definitions and Abbreviations

0.4 MAPPING BETWEEN XCP DATA TYPES AND ASAM DATA TYPES

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

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

1 THE *X*CP TRANSPORT LAYER FOR CAN

1.1 ADDRESSING

The master can use GET_SLAVE_ID to detect all *X*CP slaves within a CAN network. The master has to send GET_SLAVE_ID with the *X*CP Broadcast CAN identifier.

*X*CP on CAN uses at least two different CAN identifiers for each independent slave: one identifier for the CMD and STIM packets and one identifier for the RES, ERR, EV, SERV and DAQ packets.

The STIM CAN Identifiers may be the same as the CMD CAN Identifier or may be assigned by the SET_DAQ_ID command.

The DAQ CAN Identifiers may be the same as the RES/ERR/EV/SERV CAN Identifier or may be assigned by the SET_DAQ_ID command.

The assignment of CAN message identifiers 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. It is recommended that the bus priority of the message objects be carefully determined in order to avoid injury to other real-time communication on the bus. Also, the CMD/STIM should obtain higher priority than the RES/ERR/EV/SERV/DAQ.

The most significant bit (of the 32-bit value) set, indicates a 29 bit CAN identifier.

1.2 Communication Model

XCP on CAN makes use of the standard communication model.
The block transfer communication model is optional.
The interleaved communication model is not allowed.

1.2.1 Master Block Transfer with Incremental CAN-ID

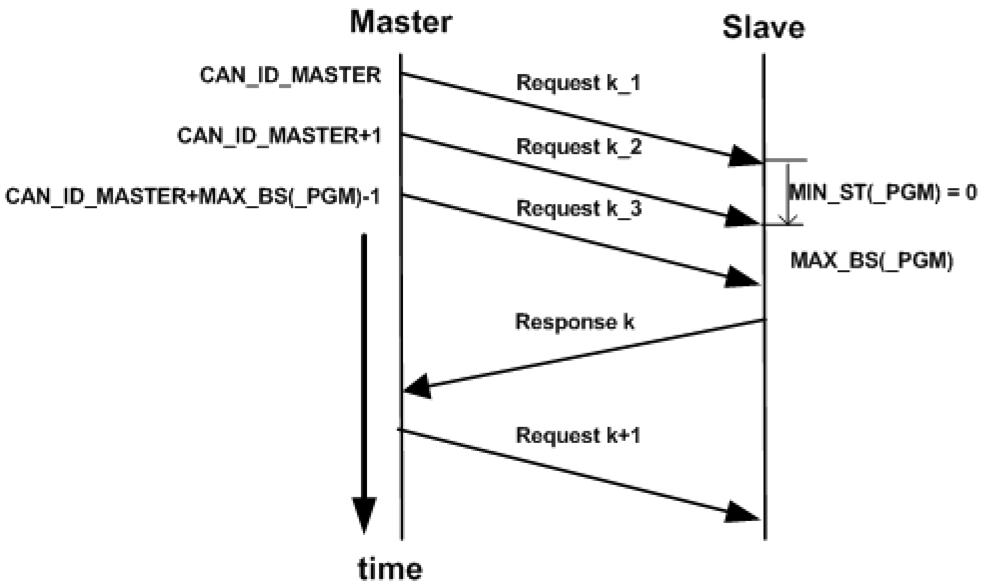


Diagram 1 : Master Block Transfer with Incremental CAN-ID

For block transfer from Master to Slave during a download or programming sequence, performance can be increased if the Master uses different CAN-ID s for every request and the communication is done with MIN_ST(_PGM) = 0.

With CAN_ID_MASTER_INCREMENTAL, the slave can inform the master that for a block transfer sequence it has to use a range of CAN-IDs for the different requests. The Master has to send the first request with CAN_ID_MASTER. The Master has to send consecutive requests by incrementing CAN_ID_MASTER for every new request. The master has to send the last request of the block transfer sequence with CAN_ID_MASTER+MAX_BS(_PGM)-1.

1.3 HEADER AND TAIL

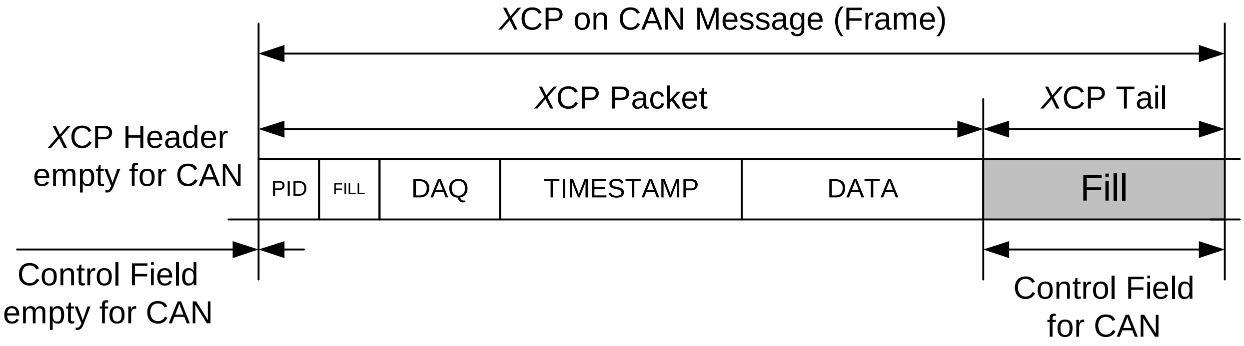


Diagram 2 : Header and Tail for *X*CP on CAN

1.3.1 HEADER

For *X*CP on CAN there's no Header (empty Control Field).

1.3.2 TAIL

For *X*CP on CAN, the Tail consists of a Control Field containing optional Fill bytes.

The maximum data length of a CAN message and therefore maximum length of an *X*CP on CAN message is MAX_DLC = 8.

If the length (LEN) of an *X*CP Packet equals MAX_DLC, the Control Field of the *X*CP Tail is empty and the *X*CP on CAN Message is the same as the *X*CP Packet (DLC = LEN = MAX_DLC).

If LEN is smaller than MAX_DLC, there're 2 possibilities to set the DLC.

A first possibility is to set DLC = LEN. The Control Field of the *X*CP Tail is empty and the *X*CP on CAN Message is the same as the *X*CP Packet.

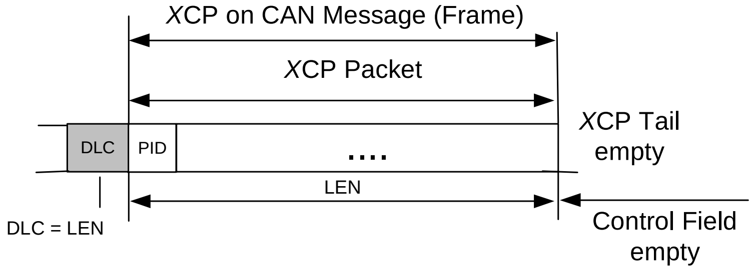


Diagram 3 : No *X*CP Tail if DLC = LEN (<= MAX_DLC)

A second possibility is to set DLC = MAX_DLC = 8. The Control Field of the XCP Tail contains MAX_DLC – LEN fill bytes. The contents of the FILL bytes is "don't care".

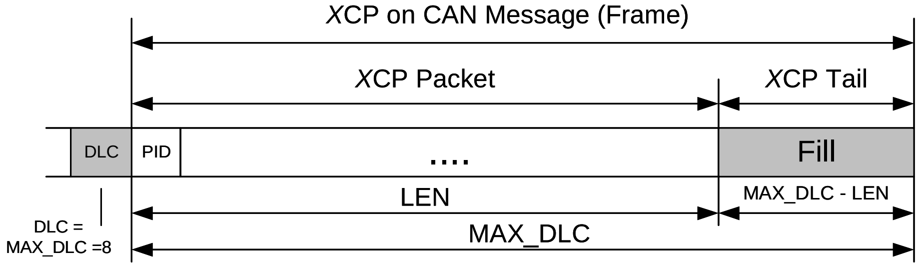


Diagram 4 : *X*CP Tail if DLC = MAX_DLC (> LEN)

With MAX_DLC_REQUIRED, the slave can inform the master that it has to use CAN frames with DLC = MAX_DLC = 8 when sending to the slave.

1.4 THE LIMITS OF PERFORMANCE

The maximum length of a CTO or a DTO packet is 8.

Name	**Type**	**Representation**	**Range of value**
MAX_CTO	Parameter	BYTE	0x08
MAX_DTO	Parameter	WORD	0x0008

2 SPECIFIC COMMANDS FOR XCP ON CAN

Table of Command Codes:

Command	Code	Timeout	Remark
GET_SLAVE_ID	0xFF	t1	optional
GET_DAQ_ID	0xFE	t1	optional
SET_DAQ_ID	0xFD	t1	optional

If SET_DAQ_ID is implemented, GET_DAQ_ID is required.

2.1 GET SLAVE CAN IDENTIFIERS

Category CAN only, optional
Mnemonic GET_SLAVE_ID

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

The master can use GET_SLAVE_ID to detect all *X*CP slaves within a CAN network.

At the same time, the master gets to know the CAN identifier the master has to use when transferring CMD/STIM to a specific slave and the CAN identifier this slave uses for transferring RES/ERR/EV/SERV/DAQ.

The master has to send GET_SLAVE_ID with the *X*CP Broadcast CAN identifier.

If the master sends an *X*CP message with the *X*CP Broadcast CAN identifier, all *X*CP slaves that are connected to the CAN network have to respond. GET_SLAVE_ID is the only *X*CP message that can be broadcasted.

A slave always has to respond to GET_SLAVE_ID, even if the slave device is not in Connected state yet.

The slave has to send the response with the CAN identifier it uses for transferring RES/ERR/EV/SERV/DAQ. The CAN identifier for CMD/STIM is coded in Intel format (MSB on higher position).

The master sends GET_SLAVE_ID with an Identification Pattern (ASCII for “XCP”). The master uses this Pattern for recognizing answers from *X*CP slaves.

If the master sends a GET_SLAVE_ID(identify by echo), the slave has to send a response that contains an echo of the Pattern. Additionally the slave informs the master about the CAN identifier the master has to use when transferring CMD/STIM to this slave.

<u>Positive Response (mode = identify by echo) :</u>

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	0x58
2	BYTE	0x43
3	BYTE	0x50
4	DWORD	CAN identifier for CMD/STIM

If the master sends a GET_SLAVE_ID(confirm by inverse echo), the slave has to send a response that contains an inversed echo of the Pattern. Additionally the slave repeats the CAN identifier the master has to use when transferring CMD/STIM to this slave.

Positive Response (mode = confirm by inversed echo) :

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	0xA7
2	BYTE	0xBC
3	BYTE	0xAF
4	DWORD	CAN identifier for CMD/STIM

If the master sends a GET_SLAVE_ID(confirm by inverse echo), without a previous GET_SLAVE_ID(identify by echo), the slaves will silently ignore that command.

If the master first sends a GET_SLAVE_ID(identify by echo) and then a GET_SLAVE_ID(confirm by inversed echo), this sequence allows the master to reliably distinguish the responses of the slaves from other communication frames on the CAN network and to reliably detect the CAN identifier pairs for every single slave.

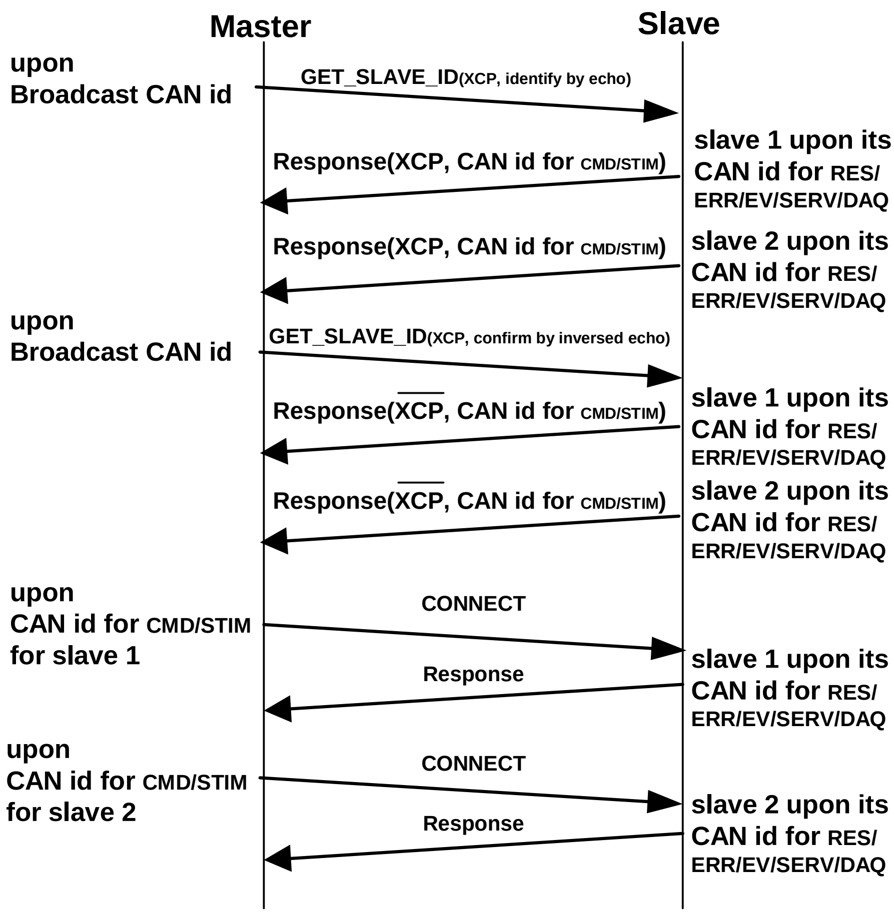


Diagram 5 : Typical use of GET_SLAVE_ID modes

2.2 GET DAQ LIST CAN IDENTIFIER

Category	CAN only, optional
Mnemonic	GET_DAQ_ID

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

Positive Response:

Position	Type	Description
0	BYTE	Packet ID: 0xFF
1	BYTE	CAN_ID_FIXED 0 = CAN-Id can be configured 1 = CAN-Id is fixed
2	WORD	Reserved
4	DWORD	CAN Identifier of DTO dedicated to list number

As a default, the master transfers all DAQ lists with DIRECTION = STIM on the same CAN Identifier as used for CMD.
Alternatively, the master may have individual CAN Identifiers (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 CAN Identifier as used for RES/ERR/EV/SERV.

Alternatively, the slave may have individual CAN Identifiers (other than the one used for RES/ERR/EV/SERV) for its DAQ lists with DIRECTION = DAQ.

With GET_DAQ_ID, the master can detect whether a DAQ list uses an individual CAN identifier and whether this Identifier is fixed or configurable.

If the CAN Identifier is configurable, the master can configure the individual Can Identifier for this DAQ list with SET_DAQ_ID.

2.3 SET DAQ LIST CAN IDENTIFIER

Category	CAN only, optional
Mnemonic	SET_DAQ_ID

Position	Type	Description
0	BYTE	Command Code = TRANSPORT_LAYER_CMD = 0xF2
1	BYTE	Sub Command Code = SET_DAQ_ID = 0xFD
2	WORD	DAQ_LIST_NUMBER [0,1,...MAX_DAQ-1]
4	DWORD	CAN Identifier of DTO dedicated to list number

The master can assign an individual CAN Identifier to a DAQ list.
If the given identifier isn't possible, the slave returns an ERR_OUT_OF_RANGE.

3 SPECIFIC EVENTS FOR *X*CP ON CAN

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

4 Interface to ASAM MCD 2MC Description File

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

4.1 ASAM MCD 2MC AML for *X*CP on CAN

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

 struct CAN_Parameters { /* At MODULE */

   uint;                                      /* XCP on CAN version */
                                              /* e.g. "1.0" = 0x0100 */

   taggedstruct {                             /* optional */
     "CAN_ID_BROADCAST"  ulong;               /* Auto detection CAN-ID            */

                                              /* master -> slaves                 */
                                              /* Bit31= 1: extended identifier    */

     "CAN_ID_MASTER"       ulong;             /* CMD/STIM CAN-ID                  */
                                              /* master -> slave                  */
                                              /* Bit31= 1: extended identifier    */
     "CAN_ID_MASTER_INCREMENTAL";             /* master uses range of CAN-IDs     */
                                              /* start of range = CAN_ID_MASTER   */
                                /* end of range = CAN_ID_MASTER+MAX_BS(_PGM)-1 */

     "CAN_ID_SLAVE"        ulong;             /* RES/ERR/EV/SERV/DAQ CAN-ID       */
                                              /* slave -> master                  */
                                              /* Bit31= 1: extended identifier    */

     "BAUDRATE"            ulong;             /* BAUDRATE [Hz] */

     "SAMPLE_POINT" uchar;                    /* sample point       */
                                              /* [% complete bit time] */
```

```
 "SAMPLE_RATE" enum {
            "SINGLE" = 1,                  /* 1 sample per bit  */
            "TRIPLE" = 3                   /* 3 samples per bit */
           };


"BTL_CYCLES" uchar;           /* BTL_CYCLES       */
                              /* [slots per bit time]   */
"SJW" uchar;                  /* length synchr. segment */
                              /* [BTL_CYCLES]           */
"SYNC_EDGE" enum {
          "SINGLE" = 1,       /* on falling edge only        */
          "DUAL"   = 2        /* on falling and rising edge */
         };

 "MAX_DLC_REQUIRED";          /* master to slave frames                */
                              /* always to have DLC = MAX_DLC = 8 */

(block "DAQ_LIST_CAN_ID" struct { /* At IF_DATA DAQ */

  uint;                       /* reference to DAQ_LIST_NUMBER */

  taggedstruct {              /* exclusive tags */
                              /* either VARIABLE or FIXED */
        "VARIABLE";
        "FIXED" ulong;        /* this DAQ_LIST always */
                              /* on this CAN_ID         */
      };

 })*;

};

};/************************* end of CAN ***********************************/
```

4.2 IF_DATA EXAMPLE FOR XCP ON CAN

```
/begin XCP_ON_CAN

  0x0100                          /* XCP on CAN version */

  CAN_ID_BROADCAST 0x0100  /* Broadcast */

  CAN_ID_MASTER       0x0200  /* CMD/STIM */
  CAN_ID_MASTER_INCREMENTAL

  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

/end XCP_ON_CAN
```

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