XCP

Version 1.0

"The Universal Measurement and Calibration Protocol Family"

Part 5

Example Communication Sequences

Association for Standardization of Automation and Measuring Systems

Dated:2003-04-08

Status of Document

Date:	2003-04-08
Authors:	Roel Schuermans, Vector Informatik GmbH Rainer Zaiser, Vector Informatik GmbH Frank Hepperle, DaimlerChrysler AG Hans Schröter, DaimlerChrysler AG Reiner Motz, Robert Bosch GmbH Andreas Aberfeld, Robert Bosch GmbH Hans-Georg Kunz, Siemens VDO Automotive AG Thomas Tyl, Siemens VDO Automotive AG Robert Leinfellner, dSPACE GmbH Hendirk Amsbeck, dSPACE GmbH Harald Styrsky, Compact Dynamics GmbH Boris Ruoff, ETAS GmbH Lars Wahlmann, Accurate Technologies Inc.
Version:	1.0
Doc-ID:	XCP -Part 5- Example Communication Sequences -1.0
Status:	Released
Type	Final

Revision History

This revision history shows only major modifications between release versions.

Date	Author	Filename	Comments
2003-04-08	R.Schuermans		Released document

Table of contents

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.

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.

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 (this document).

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 Stamp
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

1 Example Communication Sequences

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

The following notation is used for indicating the packet direction :

Symbol	Packet direction
→	Master to Slave
←	Slave to Master

1.1 Setting up a session

1.1.1 Getting basic information

Packet Type	XCP Packet	Parameters
→ CONNECT	FF 00	mode= 0x00 => NORMAL
← RES	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	
← RES	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	
← RES	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

1.1.2 Unlocking protected resources through a Seed & Key mechanism

Packet Type	*X*CP Packet	Parameters
→ GET_SEED	F8 00 01	Mode = 0x00 => first part of seed resource = 0x01 => CAL/PAG to be unlocked
← RES	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
← RES	FF 14	Current Protection Status = 0x14 => CAL/PAG unlocked, DAQ still protected, PGM still protected

Packet Type	*X*CP Packet	Parameters
→ GET_SEED	F8 00 04	Mode = 0x00 => first part of seed resource = 0x04 => DAQ to be unlocked
← RES	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
← RES	FF 10	Current Protection Status = 0x10 => CAL/PAG unlocked, DAQ unlocked, PGM still protected

Packet Type	*X*CP Packet	Parameters
→ GET_SEED	F8 00 10	Mode = 0x00 => first part of seed resource = 0x10 => PGM to be unlocked
← RES	FF 06 05 04 03 02 01 00	Mode = 0x00 => total length of seed = 0x06 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
← RES	FF 00	Current Protection Status = 0x00 => CAL/PAG unlocked, DAQ unlocked, PGM unlcoked

1.1.3 Getting information about the slave's description file

Packet Type	*X*CP Packet	Parameters
→ GET_ID	FA 01	Requested Identification Type = 0x01 => ASAM MC 2 filename without path and extension
← RES	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
← RES	FF 58 43 50 53 49 4D	Data elements in ASCII => 58 43 50 53 49 4D X C P S I M

1.2 Calibrating

1.2.1 Getting the current active pages for ECU access and XCP access

For n = 0 to MAX_SEGMENTS-1 do

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

For n = 0 to MAX_SEGMENTS-1 do

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

1.2.2 Equalizing Master and Slave through Checksum Calculation

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

1.2.3 Reading / writing slave parameters

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

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

1.2.4 Copying between pages

Packet Type	*X*CP 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
🡐 RES	FF	

1.3 Synchronous data transfer

1.3.1 Getting information about the slave's DAQ list processor

Packet Type	*XCP* Packet	Parameters
➔ GET_DAQ_PROCESSOR_INFO	DA	
← RES	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	
← RES	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

Packet Type	*XCP* Packet	Parameters
➔ GET_DAQ_EVENT_INFO	D7 xx 00 00	Event_channel_number = 0x0000 (= n)
← RES	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
← RES	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

Packet Type	*X*CP Packet	Parameters
➔ GET_DAQ_LIST_INFO	D8 xx 00 00	DAQ_list_number = 0x0000
🡐 RES	FF 04 03 0A	DAQ_LIST_PROPERTIES = 0x04 => DAQ_list_type = DAQ only MAX_ODT = 0x03 MAX_ODT_ENTRIES = 0x0A

1.3.2 Preparing the DAQ lists

1.3.2.1 Static configuration

For n = MIN_DAQ to MAX_DAQ-1 do

Packet Type	*X*CP Packet	Parameters
→ CLEAR_DAQ_LIST	E3 xx 00 00	DAQ_LIST_NUMBER = 0x0000
← RES	FF	

1.3.2.2 Dynamic configuration

Packet Type	*X*CP Packet	Parameters
➔ FREE_DAQ	D6	
← RES	FF	
➔ ALLOC_DAQ	D5 xx 01 00	DAQ_COUNT = 0x0001
← RES	FF	

For n = MIN_DAQ to MIN_DAQ+DAQ_COUNT-1 do

Packet Type	*X*CP Packet	Parameters
➔ ALLOC_ODT	D4 xx 00 00 01	DAQ_LIST_NUMBER = 0x0000 (= n) ODT_COUNT = 0x01
← RES	FF	

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

Packet Type	*X*CP Packet	Parameters
➔ ALLOC_ODT_ENTRY	D3 xx 00 00 00 02	DAQ_LIST_NUMBER = 0x0000 (= n) ODT_NUMBER = 0x00 (= i) ODT_ENTRIES_COUNT = 0x02
← RES	FF	

1.3.3 Configuring the DAQ lists

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

Packet Type	*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
← RES	FF	

For j = 0 to J_Upper_Limit do

Packet Type	*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
← RES	FF	

For the loops the following applies :

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

1.3.4 Starting the data transfer

For n = 0 to MAX_DAQ-1 do

Packet Type	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
🡐 RES	FF	

For n = 0 to MAX_DAQ-1 do

Packet Type	XCP Packet	Parameters
➔ START_STOP_DAQ_LIST	DE 02 00 00	Mode = 0x02 => select DAQ_LIST_NUMBER = 0x0000 (= n)
🡐 RES	FF	

Packet Type	XCP Packet	Parameters
➔ GET_DAQ_CLOCK	DC	
🡐 RES	FF xx xx xx AA C5 00 00	Receive timestamp = 0x0000C5AA
➔ START_STOP_SYNCH	DD 01	Mode = 0x01 => start selected
🡐 RES	FF	

1.3.5 Stopping the data transfer

For n = 0 to MAX_DAQ-1 do

Packet Type	*X*CP Packet	Parameters
→ START_STOP_DAQ_LIST	DE 02 00 00	Mode = 0x02 => select DAQ_LIST_NUMBER = 0x0000 (= n)
← RES	FF	

Packet Type	*X*CP Packet	Parameters
→ START_STOP_SYNCH	DD 02	Mode = 0x02 => stop selected
← RES	FF	

1.4 Reprogramming the slave

1.4.1 Indicating the beginning of a programming sequence

Packet Type	*X*CP Packet	Parameters
➔ PROGRAM_START	D2	
← RES	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

1.4.2 Clearing a part of non-volatile memory

Packet Type	*X*CP Packet	Parameters
➔ SET_MTA	F6 xx xx 00 00 01 00 00	Address extension = 0x00 Address = 0x00000100
← RES	FF	
➔ PROGRAM_CLEAR	D1 00 xx xx 00 01 00 00	mode= 0x00 => Absolute access mode Clear range = 0x00000100
← RES	FF	

1.4.3 Programming a non-volatile memory segment

Packet Type	*X*CP Packet	Parameters
➔ SET_MTA	F6 xx xx 00 00 01 00 00	Address extension = 0x00 Address = 0x00000100
🡐 RES	FF	

Loop with PROGRAM till end of SEGMENT

Packet Type	*X*CP Packet	Parameters
➔ PROGRAM	D0 06 00 01 02 03 04 05	Size = 0x06 Data elements = 0x00 0x01 0x02 0x03 0x04 0x05
🡐 RES	FF	

1.4.4 Indicating the end of a programming sequence

Packet Type	*X*CP Packet	Parameters
→ PROGRAM_RESET	CF	
← RES	FF	

1.5 Closing a session

Packet Type	*X*CP Packet	Parameters
➔ DISCONNECT	FE	
← RES	FF	

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

Tel.: (+49) 8102 / 895317
Fax.: (+49) 8102 / 895310
E-mail: info@asam.net
Internet: www.asam.net