ASAP Standard

CCP

CAN Calibration Protocol

Draft Version 2.1

Version	Draft 2.1
Date	23. June 1996
Status	Draft
Author	H. Kleinknecht
Distribution	

Co-Authors V2.1:

Andreas Krüger	Hewlett Packard
Hans-Georg Kunz	Mannesmann VDO
Ralf Maier	ETAS
Hans Schröter	Daimler-Benz
Rainer Zaiser	Vector Informatik

ASAP Arbeitskreis zur Standardisierung von Applikationssystemen
Standardization of Application/Calibration Systems task force

representing

AUDI AG, BMW AG, Mercedes-Benz AG, Porsche AG, Volkswagen AG and AVL List GesmbH, Robert Bosch GmbH, ETAS GmbH & Co. KG, Kleinknecht, Siemens AG, Vector Informatik GmbH, Softing GmbH, VDO AG.

Contents

1 Introduction 1
1.1 ASAP 1
1.2 CAN Calibration Protocol (CCP) 1
2 Scope and field of application 2
3 Related documents 2
4 Revision history 2
5 Definitions and Abbreviations 4
6 Protocol definition 5
6.1 Generic Control Commands 5
6.2 Data Acquisition Commands 5
7 Message Objects 6
7.1 Organisation of Message Objects 6
7.2 Description of Message Objects 7
7.2.1 Command Receive Object CRO 7
7.2.2 Data Transmission Object DTO 7
7.3 Organisation of Data Acquisition 9
8 Version Mechanism 12
9 Version compatibility 12
10 Table of Command Codes 13
11 Table of Command Return Codes 14
12 Description of Commands 15
12.1 Connect 15
12.2 Exchange Station Identifications 16
12.3 Get Seed for Key 18
12.4 Unlock Protection 19
12.5 Set Memory Transfer Address 20
12.6 Data Download 21
12.7 Data Download 6 Bytes 22
12.8 Data Upload 23
12.9 Short Upload 24
12.10 Select Calibration Data Page 25
12.11Get Size of DAQ list 26
12.12 Set DAQ list pointer 27

12.13 Write DAQ list entry....28
12.14 Start / Stop Data transmission....29
12.15 Disconnect....30
12.16 Set Session Status....31
12.17 Get Session Status....32
12.18 Build Checksum....33
12.19 Clear Memory....34
12.20 Program....35
12.21 Program 6 Bytes....36
12.22 Move memory block....37
12.23 Diagnostic Service....38
12.24 Action Service....39
12.25 Test Availability....40
12.26 Start / Stop Synchronised Data transmission....41
12.27 Get currently active Calibration Page....42
12.28 Get implemented Version of CCP....43

13 Error Handling....44

14 Example Sequences....45
14.1 Session log-in....45
14.2 Block DownLoad....45
14.3 Block UpLoad....45
14.4 Calibration Data Initialization....46
14.5 DAQ List Initialization....46
14.6 Code Update....46

15 Expected Performance Ratings....47

16 Appendices....48
16.1 Matrix of Error Codes....48
16.2 Broadcast Techniques in Multi-point Applications....49

1 Introduction

1.1 ASAP

The ASAP task force (**A**rbeitskreis zur **S**tandardisierung von **Ap**plikationssystemen; English translation: Standardization of Application/Calibration Systems task force) has been founded by the companies Audi AG, BMW AG, Mercedes-Benz AG, Porsche AG and Volkswagen AG. European manufacturers of automation, test and development systems for the automotive industry as well as manufacturers of electronic control units have joined this task force.

The world of automotive technology has grown into complex electronic system configurations to meet the increased requirements in the area of legislated exhaust gas limits, environmental pollution protection, security systems, driving performance and body equipment options. Some automotive manufacturers use in vehicle distributed control systems supported by networks.

To develop this new generation of automotive electronics, new and high sophisticated software, calibration, measurement and diagnosis equipment has to be used. At this time almost no standards exist in the area of software interfaces for such devices. Each company has its proprietary systems and interfaces to support the development of these high end configurations.

Therefore, the mission of ASAP is to reach mutual agreement and standardization in

- automation, modularisation and compatibility of all equipment to do measurement, calibration and diagnosis, and
- manage the creation of a cost reasonable and sensible tool supplier market.

1.2 CAN Calibration Protocol (CCP)

The Controller Area Network CAN is a joint development of Robert Bosch GmbH and Intel Corporation. CAN is used in many high-end automotive control systems like engine management as well as in industrial control systems. Controller chips for CAN are available from various semiconductor manufacturers.

The CAN Calibration Protocol is part of the ASAP standards. It was developed and introduced by Ingenieurbüro Helmut Kleinknecht, a manufacturer of calibration systems, and is used in several applications in the automotive industry. The CCP was taken over by the ASAP working group and enhanced with optional functions.

2 Scope and field of application

This document specifies the CAN Calibration Protocol CCP, as defined by the work group within the ASAP task force.

The CCP defines the communication of controllers with a master device using the CAN 2.0B (11-bit and 29-bit identifier), which includes 2.0A (11-bit identifier) for

1. data acquisition from the controllers,
2. memory transfers to and control functions in the controllers for calibration.

Providing these functions the CCP may be used in the areas of

- development of electronic control units (ECU)
- systems for functional and environmental tests of an ECU,
- test systems and test stands for the controlled devices (combustion engines, gearboxes, suspension systems, climatic control systems, body systems, anti-locking systems),
- on-board test and measurement systems of pre-series vehicles, and
- any non-automotive application of CAN-based distributed electronic control systems.

3 Related documents

Specifications and data sheets from Intel Corporation:

- 82527 Serial Communications Controller Datasheet (Intel #272250)
- 82527 Serial Communications Controller Architectural Overview (Intel #272410)
- Introduction to the Controller Area Network (CAN) Protocol (Intel #270962)

4 Revision history

Revision	Date	Item(s) changed	Note
1.0	30 - Sep - 1992		Initial release from Ingenieurbüro Helmut Kleinknecht
1.01b	07 - Dec - 1995	Working draft, with the following optional commands added - EXCHANGE_ID - GET_SEED - UNLOCK - GET_DAQ_SIZE - SET_DAQ_PTR - WRITE_DAQ - ACTION_SERVICE and the return / error codes - Key request - Session status request - Cold start request - Cal. data init. request - DAQ list init. request - code update request - access locked	ASAP work group

Revision	Date	Item(s) changed	Note
1.02	26 - Apr - 1996	- Explanation of DAQ lists - Extended description incl. examples - MOVE command added - PROGRAM command added	Draft for discussion in ASAP work group
2.0	14 - June - 1996	- Command STORE_CYCLE removed, - Command DNLOAD_6 added, - Command PROGRAM_6 added, - Block size in BUILD_CHKSUM, - Memory size in CLEAR_MEMORY, - Return information GET_DAQ_SIZE, - Timeout for BUILD_CHKSUM	Result of ASAP work group meeting; Document status: **Release**
Prop. 2.01	16 - March- 1998	commands modified EXCHANGE_ID : GET_SEED : START_STOP : Evt.Chnl, Prescaler BUILD_CHECKSUM :size commands added TEST GET_CCP_VERSION GET_ACT_CAL_PAGE START_STOP_ALL item removed: WCO WakeUp/Connect Object	Draft for discussion in ASAP work group Document status: **Draft**
Draft 2. 1	23 - June - 1998	Chapter **Version mechanism** added Chapter **Version compatibility** added DOWNLOAD6 classed optional SHORTUP classed optional Appendix **Matrix of Error Codes** added	Result of ASAP work group meeting; Document status: **Draft**

5 Definitions and Abbreviations

CAN

Controller Area Network : communications protocol (in ISO/OSI model level 1 + 2) developed and maintained by Robert Bosch GmbH. The protocol is designed to manage multiplexed communications between multiple CPUs. It is control information oriented, uses non-destructive bit-wise arbitration to decide, which node "owns" the bus, and has a message priority scheme based on the value of the message identifier transmitted with each message.

CCP

CAN Calibration Protocol: Developed by Ingenieurbüro Helmut Kleinknecht and adopted by the ASAP task force as a standard protocol for data acquisition and calibration.

CRO

Command Receive Object : message sent from the master device to the slave device(s).

CRM

Command Return Message : one type of message sent from the slave device to the master device containing command / error code and command counter.

DAQ

Data Acquisition : definition of a procedure and messages sent from the slave device to the master device for fast data acquisition from an ECU..

DTO

Data Transmission Object : message sent from the slave device to the master device (Command Return Message or Event Message or Data Acquisition Message).

ECU

Electronic Control Unit : an electronical device with a central processing unit performing programmed functions with its peripheral circuitry.

Message Object

A message transfered on the CAN, which is sent from a transmitting ECU to any receiving / listening ECU. The coding of the data contained in the message is "known" by the receiving ECUs. A message object's data can be 0 to 8 bytes.

Message Frame

Message Frame is a synonymous name for Message Object in the recent literature on CAN.

Master and slave devices

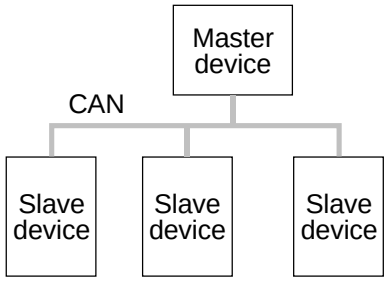
A set of controllers connected via CAN exchanges Message Objects. An additional, 'external' controller connected to the network, which communicates with one or more of these controllers and issues commands to them, here is called a master device (host). The controllers in the existing network receiving commands are called 'slave devices'.

ODT

Object Descriptor Table : list of elements (variables), used for organisation of data acquisition.

6 Protocol definition

The CAN Communication Protocol used for Calibration and Data Acquisition is a master-slave type communication. A master device is connected with one or more slave devices on the CAN:



Configuration of Master and Slave devices

The master device (host) is a calibration tool or a diagnostic / monitoring tool or a measurement system initiating the data transfer on the CAN by sending commands to the slave devices. The CCP implementation supports commands for generic control with primitive memory transfers and for data acquisition. These two parts (function sets) of the communication protocol are independant and may run asynchroneously, depending on the implementation in the slave controller. It is also possible that messages of both parts are transmitted in nested order.

6.1 Generic Control Commands

The commands are used to carry out functions with and in the slave device. For this purpose a continuous logical connection is established between the master device and any other station on CAN (slave device). This logical connection is valid until another station is selected or the current station is explicitly disconnected via command.

After the initializing logical connection has been made, data transfers from the master device to the slave device and from the slave device to the master device are controlled by the master device.

All commands from the master device have to be prompted by the selected slave device with a handshake message (command return code or error code).

6.2 Data Acquisition Commands

These protocol commands are used for continuous data acquisition from a slave device. Any CAN node may periodically transmit internal data corresponding to a list that has been configured by control commands from the master device. Data transmission is initiated by the master device and executed by the slave device and may be dependant on a fixed sampling rate and / or may be event driven (e.g. crank position).

7 Message Objects

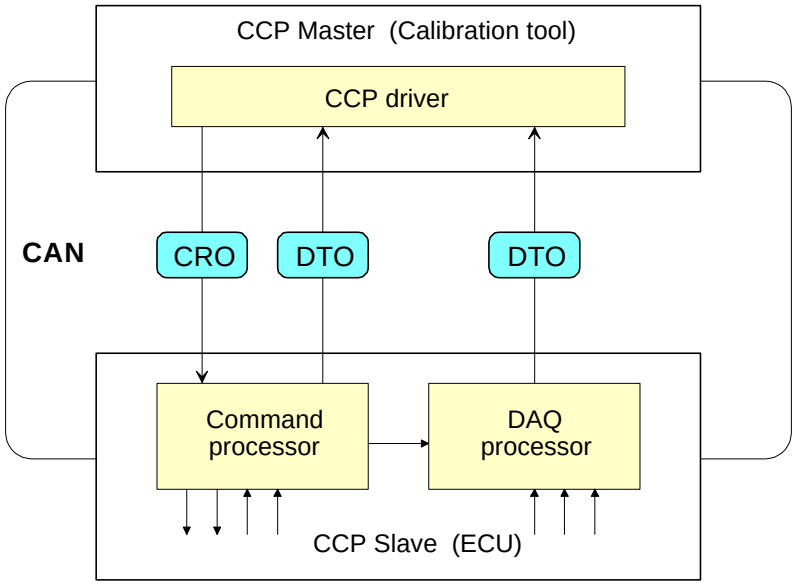
7.1 Organisation of Message Objects

According to the definition of the CAN all messages and data to be transfered are packed into "message objects", containing up to 8 byte of data. A message object is sent from one CAN node to other CAN nodes.

The CCP requires at least two message objects, one for each direction:

1) Command Receive Object : **CRO**
2) Data Transmission Object : **DTO**

The **CRO** is used for reception of command codes and related parameters to carry out internal functions or memory transfers between the logically connected CAN devices. The reception of a command has to be prompted with a handshake message using the Data Transmission Object DTO (see below), and in this case a DTO is called a **Command Return Message**. The return code of this DTO message is used to determine whether the corresponding command has been successfully completed or not.



Functional Diagram Communication flow between CCP master and slave devices.

The assignment of message identifiers to the above described objects 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 is carefully determined in order to avoid injury to other real-time communication on the bus. Also, the CRO should obtain higher priority than the DTO.

For all data transfered by the CCP **no** byte order for the protocol itself is defined. Because the data organisation depends on the ECU's CPU, the byte ordering is defined in the slave device description file. The only exceptions are the station addresses in the TEST, CONNECT and DISCONNECT commands.

7.2 Description of Message Objects

7.2.1 Command Receive Object CRO

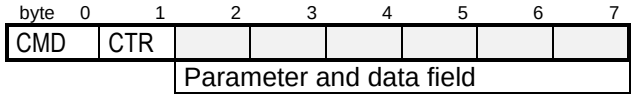
A Command Receive Object CRO is sent from the master device to one of the slave devices. The slave device answers with a Data Transmission Object DTO containing a Command Return Message CRM.

Structure of object:

Type	Rx only
Size	8 bytes message field
Purpose	Reception of commands in the slave device

Parameters in message field:

Position	Type	Description
0	byte	command code = CMD
1	byte	command counter = CTR
2...7	bytes	command related parameter and data area



The data lenght code of the CRO must always be 8. Unused data bytes, marked as „don't care„ in the command descriptions, may have arbitrary values.

7.2.2 Data Transmission Object DTO

The **DTO** has to carry all outgoing slave device messages and data as data packets. The first byte of a data packet (i.e. first byte of the DTO´s data field) is used as the **Packet ID**.

The DTO is a

- **Command Return Message** CRM, if the DTO is sent as an answer to a CRO from the master device.
- **Event Message**, if the DTO reports internal slave status changes in order to invoke error recovery or other services. Details are explained in the chapter 'Error Handling'.
- **Data Acquisition Message**, if the Packet ID points to a corresponding **Object Descriptor Table** (ODT), that describes which data acquisition elements (i.e. variables) are contained in the remaining 7 data bytes of the packet. The ODTs are initialized and modified via protocol commands (see chapter 'Description of Commands').

Structure of object:

Type	Tx (and Remote Tx Request reception)
Size	8 bytes message field
Purpose	Command Return Message CRM or Event Message or Data Acquisition Message

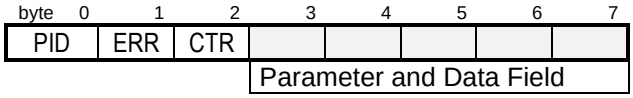
Parameters in message field:

Position	Type	Description
0	byte	data packet ID = PID
1	byte	data packet

Meaning of PID

PID	Interpretation	Note
0xFF	DTO contains a Command Return Message	
0xFE	DTO contains an Event Message	CTR is don't care
$0 \leq n \leq$ 0xFD	DTO contains a Data Acquisiton Message corresponding to ODT n	see chapter 'Descriptions of Commands'

Command Return- or Event Messages have the following format



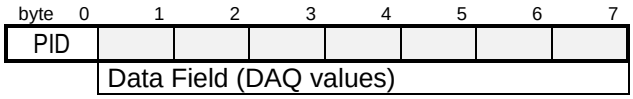
ERR: Command Return- / Error Code.

CTR: Command Counter as received in CRO with the last command.

The data lenght code of the CRM must always be 8. Unused data bytes, marked as „don't care„ in the command descriptions, may have arbitrary values.

Data Acquisiton Messages have the following format:

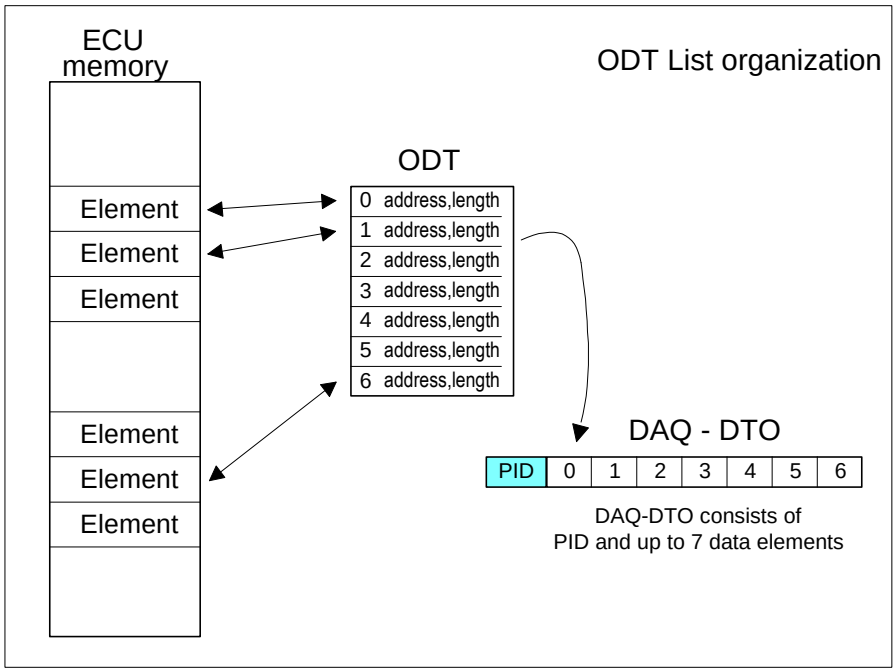
PID = n; DTO contains a Data Acquisition Message corresponding to Object Desciptor Table ODTn (explained in chapter 'Description of Commands').



The data lenght code of the DTO may be set to the actual size.

7.3 Organisation of Data Acquisition

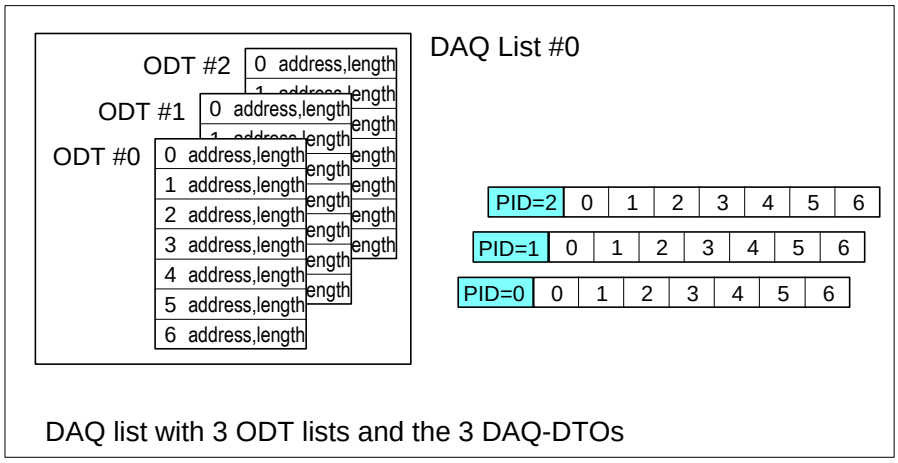
The master device can initialize data acquisition from a slave devices, which in return sends the defined data in DAQ-DTOs. The organisation of the data elements is as follows:



Data elements located in the ECU's memory are assigned to a list called **O**bject **D**escriptor **T**able ODT. This table holds the address, the address extension and the length in bytes of each element. The ODT can have up to 7 element references.

The contents of each element defined in this ODT has to be transfered into a DAQ Message DTO to be sent to the master device. To save memory resources in the ECU, the address extension and the lenght are optional. The elements have to be consistently sampled by the ECU. If a ECU does not support element sizes more than 1 byte, the master device is required to split up multibyte data object into single bytes. In this case, the ECU has to assure consistency for all elements in a ODT.

The PID is the number assigned to the ODT (0≤ n ≤ 0xFD). Typically several ODT are defined for data acquisition from a slave device:



DAQ list with 3 ODT lists and the 3 DAQ-DTOs

CCP allows the setup of a number of DAQ lists, which may be simultaneously active. The sampling and transmission of the DTOs of each DAQ list is triggered by individual events in the ECU. See the description of the START_STOP command.
When a DAQ list is triggered, the data for all or one ODTs (depending on the ECU implementation) is sampled in a consistent way. The ECU may need some time to send the sampled DTO messages on the bus. There are two possibilities for the ECU to react, when a new event cycle is triggered before the transmission of the previous cycle has finished:

1. The transmission of the previous cycle is skipped.
   Advantage: There are some ODTs of a DAQ-list missing and the tool is able to show this to the user.
   Problem: If those lacks occur frequently, there are some signals that are not measured at all.
2. The transmission of the new cycle is skipped.
   Advantage: The received samples are always complete and the resulting acquisition raster is reduced.
   Problem: In case of event-triggered samples, the tool has no possibility to check the plausibility of the acquisition-rate.

In addition, the ECU may transmit a „DAQ processor overload„ event message to inform the master device. The ECU must take care not to overflow another cycle with this additional message.
When defining the measuring component in the ECU, the developer should take care that the sampled data can be transmitted via the CAN. The described mechanisms to react on overload, should not need to be used while working in standard situation.

A typical method to set up an ODT would be:

1. Clear the current list in the ECU and let the ECU allocate memory for one DAQ list by sending the command GET_DAQ_SIZE.
   Following the GET_DAQ_SIZE command the ECU reports the available memory for ODTs in the current DAQ list.

2. In a loop send

   SET_DAQ_PTR with the parameters DAQ list number, ODT number, element number in ODT

   WRITE_DAQ with the parameters size of element, address extension, 32bit base address

   until the ODT is complete and repeat this procedure for further ODTs.

To initialize the transmission of the DAQ-DTOs a START_STOP command is issued.

8 Version Mechanism

A version management mechanism is introduced with Version 2.1 of the CCP to reduce the impact of version incompatibilites. The following policy applies:

Major protocol version number: The major protocol version number ranges from 0 to 255. A change of this dimension is necessary if the syntax and sematics of existing commands for both the memory transfer primitives and the data acquisition functions differ substantially from those defined in the previous version. This implies that non-optional commands must not be altered in a non backward compatible manner within the same major protocol version number.

Minor protocol version number: The minor protocol version number ranges from 0 to 9. A change of this dimension is necessary if new optional commands are added / removed to / from the protocol.

9 Version compatibility

To ensure a minimum compatibility between the master and the slave device the following policies apply:

1. Communication between a master and a slave device is not possible if the major protocol version numbers are not equal.

2. The master device must have at least have the same protocol version (major and minor protocol version number) or higher than the slave device. The slave device is only required to support one protocol version.

3. In order to ensure a full ASAP1a CCP conformity the master and the slave device must at least support all non-optional commands.

10 Table of Command Codes

Command	Code	TimeOut to ACK [ms]	Remark
CONNECT	0x01	25	
GET_CCP_VERSION	0x1B	25	
EXCHANGE_ID	0x17	25	
GET_SEED	0x12	25	optional command
UNLOCK	0x13	25	optional command
SET_MTA	0x02	25	
DNLOAD	0x03	25	
DNLOAD_6	0x23	25	optional command
UPLOAD	0x04	25	
SHORT_UP	0x0F	25	optional command
SELECT_CAL_PAGE	0x11	25	optional command
GET_DAQ_SIZE	0x14	25	
SET_DAQ_PTR	0x15	25	
WRITE_DAQ	0x16	25	
START_STOP	0x06	25	
DISCONNECT	0x07	25	
SET_S_STATUS	0x0C	25	optional command
GET_S_STATUS	0x0D	25	optional command
BUILD_CHKSUM	0x0E	30 000	optional command
CLEAR_MEMORY	0x10	30 000	optional command
PROGRAM	0x18	100	optional command
PROGRAM_6	0x22	100	optional command
MOVE	0x19	30 000	optional command
TEST	0x05	25	optional command
GET_ACTIVE_CAL_PAGE	0x09	25	optional command
START_STOP_ALL	0x08	25	optional command
DIAG_SERVICE	0x20	500	optional command
ACTION_SERVICE	0x21	5 000	optional command

The following comands are optional, if the ECU does not support DAQ:

GET_DAQ_SIZE,
SET_DAQ_PTR,

WRITE_DAQ,

START_STOP.

If SELECT_CAL_PAGE is implemented, GET_ACTIVE_CAL_PAGE is required.

11 Table of Command Return Codes

Code	Description	Error category	State transition to	
0x00	acknowledge / no error	-		
0x01	DAQ processor overload	C0		
0x10	command processor busy	C1	NONE (wait until ACK or timeout)	
0x11	DAQ processor busy	C1	NONE (wait until ACK or timeout)	
0x12	internal timeout	C1	NONE (wait until ACK or timeout)	
0x18	key request	C1	NONE (embedded seed&key)	
0x19	session status request	C1	NONE (embedded SET_S_STATUS)	
0x20	cold start request	C2	COLD START	
0x21	cal. data init. request	C2	cal. data initialization	
0x22	DAQ list init. request	C2	DAQ list initialization	
0x23	code update request	C2	(COLD START)	
0x30	unknown command	C3	(FAULT)	
0x31	command syntax	C3	FAULT	
0x32	parameter(s) out of range	C3	FAULT	
0x33	access denied	C3	FAULT	
0x34	overload	C3	FAULT	
0x35	access locked	C3	FAULT	
0x36	resource/function not available	C3	FAULT	

If errors occur asynchroneously to protocol commands, the CCP slave device may also directly invoke appropriate error handling by sending error codes as ***Event Messages*** (Packet ID 0xFE).

Categories of errors:

Category	Description	Action	Retries
timeout	no handshake message	retry	2
C0	warning	-	-
C1	spurious (comm error, busy, ...)	wait (ACK or timeout)	2
C2	resolvable (temp. power loss, ...)	reinitialize	1
C3	unresolvable (setup, overload, ...)	terminate	-

For further details see Chapter ‘Error Handling’.

12 Description of Commands

In this chapter all commands with their parameters and the expected return information are explained, including an example for each command.

12.1 Connect

Command label **CONNECT**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = CONNECT 0x01
1	byte	Command Counter = CTR
2	word	station address (Intel format)
4...7	bytes	don't care

This command establishes a continuous ***logical point-to-point connection*** with the selected slave station for the master-slave command protocol. All following protocol commands refer to this station only, until another station is selected. A CONNECT command to another station temporary disconnects the active station (See DISCONNECT). A CONNECT command to an already connected station is acknowledged. A slave device does not respond to any commands unless being addressed by a prior CONNECT command with the correct station address The ***station address*** is specified as a number in little-endian byte order (Intel format, low byte first).

The following return information is expected (contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3...7	bytes	don't care

Example

The master device sends a CONNECT CRO to the slave device with the station address 0x0200. The command counter CTR currently is 0x45:

byte 0	1	2	3	4	5	6	7
0x01	0x45	0x00	0x02	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x45	--	--	--	--	--

12.2 Exchange Station Identifications

Command label **EXCHANGE_ID**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = EXCHANGE_ID 0x17
1	byte	Command Counter = CTR
2...	bytes	CCP master device ID information (optional and implementation specific)

The CCP master and slave stations exchange IDs for automatic session configuration. This might include automatic assignment of a data acquisition setup file depending on the slave's returned ID (Plug´n´Play). The following return information is expected (contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	length of slave device ID in bytes
4	byte	data type qualifier of slave device ID (optional and implementation specific)
5	byte	Resource Availability Mask
6	byte	Resource Protection Mask
7	byte	Don’t care

The slave device automatically sets the ***Memory Transfer Address*** 0 (MTA0) to the location from which the CCP master may subsequently upload the requested ID using UPLOAD. See also the SET_MTA and UPLOAD command description.

Resource Availabilty Mask / Resource Protection Status Mask format for simplification in the following just referred to as resource mask.

Bit 7	6	5	4	3	2	1	0
x	**PGM**	**x**	**x**	**X**	**X**	**DAQ**	**CAL**

CAL	Calibration
DAQ	Data Acquisition
PGM	Memory programming
x	reserved for future use

Resource Availability: if bit=TRUE specified resource or function is available.

Resource Protection: if bit=TRUE specified resource or function is protected against unauthorized access (needs UNLOCK).

Example

The master device sends a EXCHANGE_ID CRO to the slave device. The command counter CTR currently is 0x23:

byte 0	1	2	3	4	5	6	7
0x17	0x23	--	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the length of the slave ID and its data type:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x04	0x02	0x03	0x03	--

The length of the slave ID is 4 bytes, the data type is coded as type 2. The resource availabitity mask is 0x03, the resource protection status mask is 0x03. The data can be fetched from the slave using UPLOAD.

12.3 Get Seed for Key

Command label **GET_SEED**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = GET_SEED 0x12
1	byte	Command Counter = CTR
2	byte	Requested slave resource or function (Resource Mask)
3...7	bytes	don't care

See EXCHANGE_ID for a description of the resource mask.

Only one resource or function may be requested with one GET_SEED command If more than one resource is requested, the GET_SEED command together with the following UNLOCK command has to be performed multiple times.

Returns ´seed´ data for a ***seed&key*** algorithm for computing the ´key´ to unlock the requested function for authorized access (see 'Unlock Protection' below).

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	Protection status (TRUE or FALSE)
4...7	bytes	'seed' data

If ***Protection status*** = FALSE, no UNLOCK is required to unlock the requested function.

Example

The master device sends a GET_SEED CRO to the slave device. The command counter CTR currently is 0x23. The task is data acquisition :

byte 0	1	2	3	4	5	6	7
0x12	0x23	0x02	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the protection status and the 'seed' data:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x01	0x14	0x15	0x16	0x17

The protection status is TRUE (0x01), the seed data are 0x14, 0x15, 0x16, 0x17.

12.4 Unlock Protection

Command label **UNLOCK**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = UNLOCK 0x13
1	byte	Command Counter = CTR
2...	bytes	'key'

Unlocks the slave devices security protection (if applicable) using a ´key´ computed from ´seed´. See *seed&key* above.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	Current Privilege Status (Resource Mask)
4...7	bytes	don't care

See EXCHANGE_ID for a description of the resource mask.

Example

The master device sends an UNLOCK CRO to the slave device. The command counter CTR currently is 0x23 and the key found with GET_SEED is used:

byte 0	1	2	3	4	5	6	7
0x13	0x23	0x14	0x15	0x16	0x17	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, and the privilege status:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x02	--	--	--	--

The privilege status is 0x02, only Data Acquisition is unlocked.

12.5 Set Memory Transfer Address

Command label **SET_MTA**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = SET_MTA 0x02
1	byte	Command Counter = CTR
2	byte	Memory transfer address MTA number (0,1)
3	byte	Address extension
4..7	unsigned long	Address

This command will initialize a base pointer (32Bit + extension) for all following memory transfers. The address extension depends on the slave controller's organization and may identify a switchable memory bank or a memory segment.

The MTA number (handle) is used to identify different transfer address locations (pointers). MTA0 is used by the commands DNLOAD, UPLOAD, DNLOAD_6 , SELECT_CAL_PAGE, CLEAR_MEMORY, PROGRAM and PROGRAM_6. MTA1 is used by the MOVE command. See also command 'MOVE'.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3...7	bytes	don't care

Example

The master device sends a SET_MTA CRO to the slave device. The command counter CTR currently is 0x23, the MTA0 number is 0, the address extension is 0x02 and the base address is 0x34002000:

byte 0	1	2	3	4	5	6	7
0x02	0x23	0x00	0x02	0x34	0x00	0x20	0x00

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.6 Data Download

Command label **DNLOAD**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = DNLOAD 0x03
1	byte	Command Counter = CTR
2	byte	size of data block to follow in bytes
3...7	bytes	data to be transferred (up to 5 bytes)

The data block of the specified length (size) contained in the CRO will be copied into memory, starting at the current Memory Transfer Address0 (MTA0). The MTA0 pointer will be post-incremented by the value of 'size'.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	MTA0 extension (after post-increment)
4...7	unsigned long	MTA0 address (after post-increment)

Example

The master device sends a DNLOAD CRO to the slave device. The command counter CTR currently is 0x23, the size of data is 0x05 and the data to be transferred is 0x10, 0x11, 0x12, 0x13, 0x14:

byte	0	1	2	3	4	5	6	7
	0x03	0x23	0x05	0x10	0x11	0x12	0x13	0x14

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the memory transfer address extension 0x02 and the (incremented) current MTA0

byte	0	1	2	3	4	5	6	7
	0xFF	0x00	0x23	0x02	0x34	0x00	0x20	0x05

Before execution of this command the MTA0 had been 0x34002000 and was incremented by 5.

12.7 Data Download 6 Bytes

Command label **DNLOAD_6**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = DNLOAD_6 0x23
1	byte	Command Counter = CTR
2...7	bytes	6 bytes of data to be transferred

The data block with the fixed length (size) of 6 bytes contained in the CRO will be copied into memory, starting at the current Memory Transfer Address0 (MTA0). The MTA0 pointer will be post-incremented by the value 6.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	MTA0 extension (after post-increment)
4..7	unsigned long	MTA0 address (after post-increment)

Example

The master device sends a DNLOAD_6 CRO to the slave device. The command counter CTR currently is 0x25, the size of data is 0x05 and the data to be transferred is 0x10, 0x11, 0x12, 0x13, 0x14, 0x15:

byte 0	1	2	3	4	5	6	7
0x23	0x25	0x10	0x11	0x12	0x13	0x14	0x15

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the memory transfer address extension 0x02 and the (incremented) current MTA0

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x25	0x02	0x34	0x00	0x20	0x06

Before execution of this command the MTA0 had been 0x34002000 and was incremented by 6.

12.8 Data Upload

Command label **UPLOAD**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = UPLOAD 0x04
1	byte	Command Counter = CTR
2	byte	Size of data block to be uploaded in bytes
3...7	bytes	don't care

A data block of the specified length (size), starting at current MTA0, will be copied into the corresponding DTO data field. The MTA0 pointer will be post-incremented by the value of 'size'.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	requested data bytes

Example

The master device sends an UPLOAD CRO to the slave device. The command counter CTR currently is 0x23, the size of data is 0x04:

byte 0	1	2	3	4	5	6	7
0x04	0x23	0x04	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO and the requested data bytes:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x10	0x11	0x12	0x13	--

The new MTA0 address is not reported in the DTO.

12.9 Short Upload

Command label **SHORT_UP**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = SHORT_UP 0x0F
1	byte	Command Counter = CTR
2	byte	Size of data block to be uploaded in bytes (1...5)
3	byte	Address extension
4	unsigned long	Address

A data block of the specified length (size), starting at source address will be copied into the corresponding DTO data field. The MTA0 pointer remains unchanged.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	requested data bytes

Example

The master device sends a SHORT_UP CRO to the slave device. The command counter CTR currently is 0x23, the size of data is 0x04 and the source adress is 0x12345678:

byte 0	1	2	3	4	5	6	7
0x0F	0x23	0x04	0x00	0x12	0x34	0x56	0x78

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO and the requested data bytes:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x10	0x11	0x12	0x13	--

12.10 Select Calibration Data Page

Command label **SELECT_CAL_PAGE**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = SELECT_CAL_PAGE 0x11
1	byte	Command Counter = CTR
2...7	bytes	don't care

This command's function depends on the ECU implementation. The previously initialized MTA0 points to the start of the calibration data page that is selected as the currently active page by this command.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code = ACKNOWLEDGE 0x00
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends first a SET_MTA CRO and now a SELECT_CAL_PAGE CRO to the slave device. The command counter CTR currently is 0x23.

byte 0	1	2	3	4	5	6	7
0x11	0x23	--	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

Using two blocks of ECU memory for calibration, the SET_MTA and the SELECT_CAL_PAGE commands can be used for a kind of 'emergency interrupt' from the master device in order to bring the slave system into a 'safe state' by preparing the change of these two memory blocks with the SET_MTA command and executing the change immediately with SELECT_CAL_PAGE.

12.11 Get Size of DAQ list

Command label **GET_DAQ_SIZE**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = GET_DAQ_SIZE 0x14
1	byte	Command Counter = CTR
2	byte	DAQ list number (0,1,...)
3	byte	don't care
4..7	unsigned long	CAN Identifier of DTO dedicated to list number

Returns the size of the specified DAQ list as the number of available Object DescriptorTables (ODTs) and clears the current list. If the specified list number is not available, size = 0 should be returned. The DAQ list is initialized and data acquisition by this list is stopped.

An individual CAN Identifier may be assigned to a DAQ list to configure multi ECU data acquisition. This feature is optional. If the given identifier isn't possible, an error code is returned. 29 bit CAN identifiers are marked by the most significant bit set.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	DAQ list size (= number of ODTs in this list)
4	byte	First PID of DAQ list
5 ... 7	bytes	don't care

The PID of a specific ODT of a DAQ list is determined by:

PID = First PID of DAQlist + ODT number

Example

The master device sends a GET_DAQ_SIZE CRO to the slave device. The command counter CTR currently is 0x23, the DAQ list number is 0x03 with the ID 0x01020304.

byte 0	1	2	3	4	5	6	7
0x14	0x23	0x03	--	0x01	0x02	0x03	0x04

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the first PID=0x08 and the list size 0x10 (10 ODT with up to 7 elements each)

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x10	0x08	--	--	--

12.12 Set DAQ list pointer

Command label **SET_DAQ_PTR**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = SET_DAQ_PTR 0x15
1	byte	Command Counter = CTR
2	byte	DAQ list number (0,1,...)
3	byte	Object Descriptor Table ODT number (0,1,...)
4	byte	Element number within ODT (0,1,...)
5...7	bytes	don't care

Initializes the DAQ list pointer for a subsequent write to a DAQ list.
See also 'Organization of Data Acquisition Messages'.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends a SET_DAQ_PTR CRO to the slave device. The command counter CTR currently is 0x23, the DAQ list number is 0x03, the ODT number is 0x05 and the element number addressed is 0x02.

byte 0	1	2	3	4	5	6	7
0x15	0x23	0x03	0x05	0x02	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

Next the command WRITE_DAQ is used to set the data elements in the selected ODT.

12.13 Write DAQ list entry

Command label **WRITE_DAQ**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = WRITE_DAQ 0x16
1	byte	Command Counter = CTR
2	byte	Size of DAQ element in bytes { 1, 2, 4 }
3	byte	Address extension of DAQ element
4...7	unsigned long	Address of DAQ element

Writes one entry (description of single DAQ element) to a DAQ list defined by the DAQ list pointer (see SET_DAQ_PTR). The following DAQ element sizes are defined: 1 byte , 2 bytes (type word), 4 bytes (type long / Float).

An ECU may not support individual address extensions for each element and 2 or 4 byte element sizes. It is the responsibility of the master device to care for the ECU limitations. The limitations may be defined in the slave device description file (e.g. ASAP2).

It is the responsibility of the slave device, that all bytes of a DAQ element are consistent upon transmission.

See also 'Organization of Data Acquisition Messages'.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends an WRITE_DAQ CRO to the slave device. The command counter CTR currently is 0x23, the size of the DAQ element is 0x02 bytes, the address extension is 0x01 and the 32-bit address is 0x02004200.

byte 0	1	2	3	4	5	6	7
0x16	0x23	0x02	0x01	0x02	0x00	0x42	0x00

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.14 Start / Stop Data transmission

Command label **START_STOP**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = START_STOP 0x06
1	byte	Command Counter = CTR
2	byte	Mode : start / stop / prepare data tranmission
3	byte	DAQ list number
4	byte	Last ODT number
5	byte	Event Channel No.
6, 7	word	Transmission rate prescaler

This command is used to start or to stop the data acquisition or to prepare a synchronized start of the specified DAQ list. The ***Last ODT number*** specifies which ODTs (From 0 to Last ODT number) of this DAQ list should be transmitted. The ***Event Channel No***. specifies the generic signal source that effectively determines the data transmission timing To allow reduction of the desired transmission rate, a ***prescaler*** may be applied to the Event Channel. The prescaler value factor must be greater than or equal to 1.
The ***mode*** parameter is defined as follows: 0x00 stops specified DAQ list, 0x01 starts specified DAQ list, 0x02 prepares DAQ list for synchronised start.
The start/stop mode parameter = 0x02 (prepare to start data transmission) configures the DAQ list with the provided parameters but does not start the data acquisition of the specified list. This parameter is used for a synchronized start of all configured DAQ lists. In case the slave device is not capable of performing the synchronized start of the data acquisition, the slave device may then start data transmission if this parameter is TRUE (not zero).
The ECU specific properties of event channels and DAQ lists may be described in the slave device description (ASAP2).

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends a START_STOP CRO to the slave device. The command counter CTR currently is 0x23, the start/stop byte is 0x01 (start), the DAQ list number is 0x03and the packet number to transmit is 0x07. The ECU shell use the event-channel 0x02 with a prescaler of 1 (motorola-format).

byte 0	1	2	3	4	5	6	7
0x06	0x23	0x01	0x03	0x07	0x02	0x00	0x01

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.15 Disconnect

Command label **DISCONNECT**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = DISCONNECT 0x07
1	byte	Command Counter = CTR
2	byte	0x00 temporary, 0x01 end of session
3	byte	don't care
4,5	word	Station address (Intel format)
6...7	bytes	don't care

Disconnects the slave device. The disconnect can be temporary, setting the slave device in an "off line" state or with parameter 0x01 terminating the calibration session.

Terminating the session invalidates all state information and resets the slave protection status.

A temporary disconnect doesn't stop the transmission of DAQ messages The MTA values, the DAQ setup, the session status and the protection status are unaffected by the temporary disconnect and remain unchanged.

If the ECU supports the resume feature and the resume bit was set with a SET_SESSION_STATUS command, the DAQ related functions behave like in a temporary disconnect. The protections status for DAQ remains unlocked.

Station address is specified as a number in little-endian byte order (Intel format, low byte first).

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends a DISCONNECT CRO to the slave device. The command counter CTR currently is 0x23, the parameter byte is 0x00 (temporary) and the slave device address is 0x0208.

byte 0	1	2	3	4	5	6	7
0x07	0x23	0x00	--	0x08	0x02	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.16 Set Session Status

Command label **SET_S_STATUS**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = SET_S_STATUS 0x0C
1	byte	Command Counter = CTR
2	byte	Session status bits (see table below)
3...7	byte	don't care

Keeps the slave node informed about the current state of the calibration session (see also chapter 'Error Handling').

Session Status:

Bit 7	6	5	4	3	2	1	0
RUN	STORE	res	res	res	RESUME	DAQ	CAL

bit 0	CAL	Calibration data initialized
bit 1	DAQ	DAQ list(s) initialized
bit 2	RESUME	Request to save DAQ setup during shutdown in ECU. ECU automatically restarts DAQ after startup.
bit 6	STORE	Request to save calibration data during shut-down in ECU
Bit 7	RUN	Session in progress
bit 3..5	res	reserved

Bits are set (1) if expression is TRUE.

The session status bits are read/write to the slave device andare be cleared on power-up, on session log-off and in applicable fault conditions.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends a SET_S_STATUS CRO to the slave device. The command counter CTR currently is 0x23, the session status bits are 10000001 (RUN, CAL).

byte 0	1	2	3	4	5	6	7
0x0C	0x23	0x81	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.17 Get Session Status

Command label **GET_S_STATUS**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = GET_S_STATUS 0x0D
1	byte	Command Counter = CTR
2...7	bytes	don't care

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID = 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	Session status
4	byte	additional status information qualifier
5	bytes	additional status information (**optional**)

Note: the use of ***additional status information*** is manufacturer and / or project specific, it is not part of this protocol specification. For example, additional status information could contain an incremental checksum result, that keeps track of the current session activities.
If the return information does not contain additional status information, the ***additional status information qualifier*** has to be FALSE (0). If the additional status information is not FALSE, it may be used to determine the type of additional status information.

Example

The master device sends a GET_S_STATUS CRO to the slave device. The command counter CTR currently is 0x23.

byte 0	1	2	3	4	5	6	7
0x0D	0x23	--	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO and the session status bits:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x81	--	--	--	--

The session status is bit 0 set (CAL) and bit 7 set (RUN).

12.18 Build Checksum

Command label **BUILD_CHKSUM**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = BUILD_CHKSUM 0x0E
1	byte	Command Counter = CTR
2..5	unsigned long	block size in bytes
6, 7	bytes	don't care

Returns a checksum result of the memory block that is defined by MTA0 (memory transfer area start address) and block ***size***. The checksum algorithm may be manufacturer and / or project specific, it is not part of this specification.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	➲ size of checksum data
4 ... 7	bytes	➲ checksum data (implementation specific)

Example

The master device sends a BUILD_CHKSUM CRO to the slave device. The command counter CTR currently is 0x23, the block size is 32 kbytes (0x8000).

byte 0	1	2	3	4	5	6	7
0x0E	0x23	0x00	0x00	0x80	0x00	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO and the calculated checksum 0x1234.

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x02	0x12	0x34	--	--

12.19 Clear Memory

Command label **CLEAR_MEMORY**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = CLEAR_MEMORY 0x10
1	byte	Command Counter = CTR
2..5	long	Memory size
3...7	bytes	don't care

This command may be used to erase FLASH EPROM prior to reprogramming.The MTA0 pointer points to the memory location to be erased.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends a CLEAR_MEMORY CRO to the slave device. The command counter CTR currently is 0x23, the block size is 32 kbytes (0x8000).

byte 0	1	2	3	4	5	6	7
0x10	0x23	0x00	0x00	0x80	0x00	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.20 Program

Command label **PROGRAM**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = PROGRAM 0x18
1	byte	Command Counter = CTR
2	byte	Size of data block to follow (bytes)
3...7	bytes	Data to be programmed (max. 5 bytes)

The data block of the specified length (size) contained in the CRO will be programmed into non-volatile memory (FLASH, EEPROM), starting at current MTA0. The MTA0 pointer will be post-incremented by the value of ‘size’.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	MTA0 extension (after post-increment)
4	unsigned long	MTA0 address (after post-increment)

Example

The master device sends a PROGRAM command to the slave device. The command counter CTR currently is 0x23, the size is 3 bytes (0x03) and the data to be programmed is 0x10, 0x11, 0x12:

byte 0	1	2	3	4	5	6	7
0x18	0x23	0x03	0x10	0x11	0x12	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO, the memory transfer address extension 0x02 and the (incremented) current MTA0

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x02	0x34	0x00	0x20	0x03

Before execution of this command the MTA0 had been 0x34002000 and was incremented by 3.

12.21 Program 6 Bytes

Command label **PROGRAM_6**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = PROGRAM_6 0x22
1	byte	Command Counter = CTR
2...7	bytes	Data to be programmed (6 bytes)

The data block with the length (size) of 6 bytes contained in the CRO will be programmed into non-volatile memory (FLASH, EEPROM), starting at current MTA0. The MTA0 pointer will be post-incremented by 6.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	MTA0 extension (after post-increment)
4	unsigned long	MTA0 address (after post-increment)

Example

The master device sends a PROGRAM command to the slave device. The command counter CTR currently is 0x23, the size is 6 bytes and the data to be programmed is 0x10, 0x11, 0x12, 0x13, 0x14, 0x15:

byte 0	1	2	3	4	5	6	7
0x22	0x23	0x10	0x11	0x12	0x13	0x14	0x15

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO, the memory transfer address extension 0x02 and the (incremented) current MTA0

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x02	0x34	0x00	0x20	0x06

Before execution of this command the MTA0 had been 0x34002000 and was incremented by 6.

12.22 Move memory block

Command label **MOVE**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = MOVE 0x19
1	byte	Command Counter = CTR
2..5	long	Size (number of bytes) of data block to be moved
6,7	bytes	don't care

A data block of the specified length (size) will be copied from the address defined by MTA 0 (source pointer) to the address defined by MTA 1 (destination pointer).

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

Example

The master device sends a MOVE CRO to the slave device. The command counter CTR currently is 0x23, the number of bytes to move is 32 kbytes (0x8000).

byte 0	1	2	3	4	5	6	7
0x19	0x23	0x00	0x00	0x80	0x00	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00) and the CTR of the CRO:

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	--	--	--	--	--

12.23 Diagnostic Service

Command label **DIAG_SERVICE**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = DIAG_SERVICE 0x20
1	byte	Command Counter = CTR
2,3	word	Diagnostic service number
4...7	bytes	Parameters, if applicable

The slave device carries out the requested service and automatically sets the Memory Transfer Address MTA0 to the location from which the CCP master device (host) may subsequently upload the requested diagnostic service return information.

The following handshake is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID = 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	length of return information in bytes
4	byte	data type qualifier of return information (to be defined)
5...7	bytes	don't care

Example

The master device sends a DIAG_SERVICE CRO to the slave device. The command counter CTR currently is 0x23, the requested diagnostic service number is 0x08 and has no parameters.

byte 0	1	2	3	4	5	6	7
0x20	0x23	0x08	--	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the length of return information 0x20 and the data type 0x00.

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x20	0x00	--	--	--

The diagnostic service data now can be uploaded from the MTA0. See UPLOAD.

12.24 Action Service

Command label **ACTION_SERVICE**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = ACTION_SERVICE 0x21
1	byte	Command Counter = CTR
2,3	word	Action service number
4...7	bytes	Parameters, if applicable

The slave device carries out the requested service and automatically sets the Memory Transfer Address MTA0 to the location from which the CCP master device may subsequently upload the requested action service return information (if applicable).

The following handshake is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID = 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3	byte	length of return information in bytes
4	byte	data type qualifier of return information (to be defined)
5...7	bytes	don't care

Example

The master device sends an ACTION_SERVICE CRO to the slave device. The command counter CTR currently is 0x23, the requested action service number is 0x08 and has the parameter 0x05.

byte 0	1	2	3	4	5	6	7
0x20	0x23	0x08	0x05	--	--	--	--

The slave device answers with a DTO containing ACKNOWLEDGE (0x00), the CTR of the CRO, the length of return information 0x20 and the data type 0x00.

byte 0	1	2	3	4	5	6	7
0xFF	0x00	0x23	0x20	0x00	--	--	--

The resulting action service data now can be uploaded from the MTA0. See UPLOAD.

12.25 Test Availability

Command label **TEST**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = TEST 0x05
1	byte	Command Counter = CTR
2,3	little-endian word	station address (Intel format)
4...	bytes	don't care

This command is used to test if the slave with the specified station address is available for CCP communication. This command does not establish a logical connection nor does it trigger any activities in the specified slave. ***Station address*** is specified as a number in little-endian byte order (Intel format, low byte first).

The following return information is expected (contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code = ACK 0x00
2	byte	Command Counter = CTR
3 ...7	bytes	don't care

12.26 Start / Stop Synchronised Data transmission

Command label **START_STOP_ALL**

Structure of data in CRO

Position	Type	Description
0	byte	Command Code = START_STOP 0x08
1	byte	Command Counter = CTR
2	byte	0x00 stops, 0x01 starts data transmission
3 ... 7	bytes	don´t care

This command is used to start the periodic transmission of all DAQ lists configured with the previously sent START_STOP command (start/stop modus = 2) as „prepared to start“ in a synchronized manner. The command is used to stop the periodic transmission of **all** DAQ lists, including those not started synchronized.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	byte	Packet ID: 0xFF
1	byte	Command Return Code
2	byte	Command Counter = CTR
3 ... 7	bytes	don't care

12.27 Get currently active Calibration Page

Command label **GET_ACTIVE_CAL_PAGE**

Structure of data in CRO

Position	Type	Description
0	Byte	Command Code = GET_ACTIVE_CAL_PAGE 0x09
1	Byte	Command Counter = CTR
2...7	Bytes	don't care

This command returns the start address of the calibration page that is currently active in the slave device.

The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	Byte	Packet ID: 0xFF
1	Byte	Command Return Code
2	Byte	Command Counter = CTR
3	Byte	Address extension
4 ... 7	unsigned long	Address

12.28 Get implemented Version of CCP

Command label **GET_CCP_VERSION**

Structure of data in CRO

Position	Type	Description
0	Byte	Command Code = GET_CCP_VERSION 0x1B
1	Byte	Command Counter = CTR
2	Byte	Main Protocol version (desired)
3	Byte	Release within version (desired)
4 ...7	Bytes	don't care

This command performs a mutual identification of the protocol version used in the master and in the slave device to aggree on a common protocol version. This command is expected to be executed prior to the EXCHANGE_ID command.

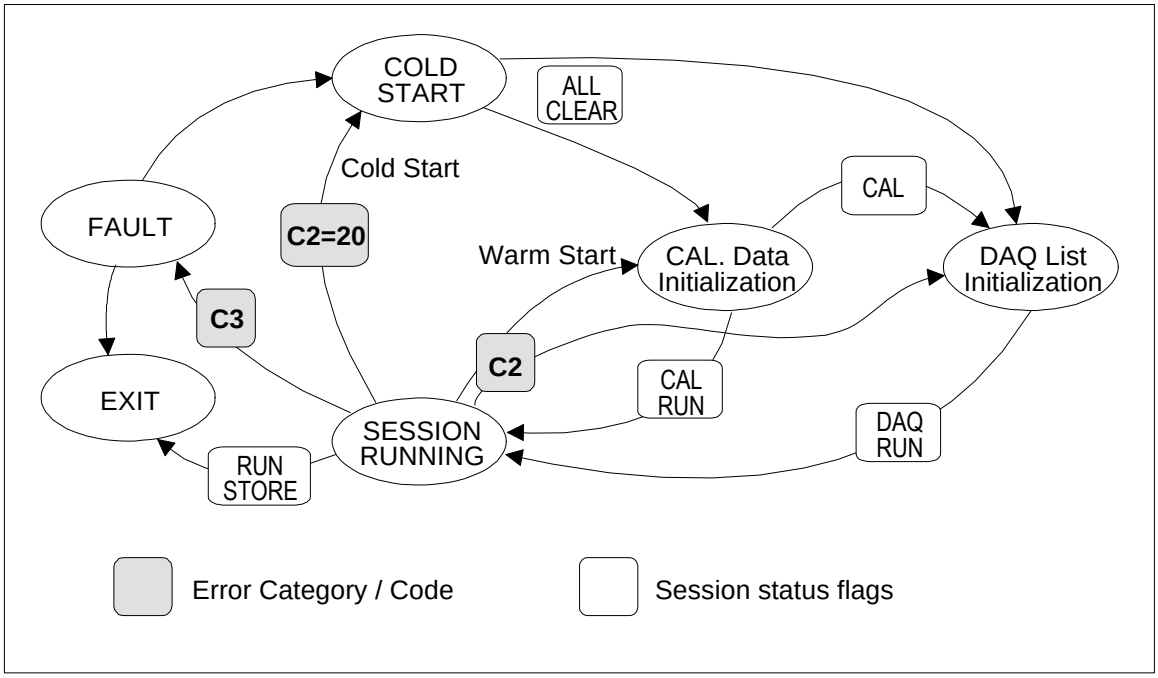
The following return information is expected (Contents of returned DTO:)

Position	Type	Description
0	Byte	Packet ID: 0xFF
1	Byte	Command Return Code
2	Byte	Command Counter = CTR
3	Byte	Main Protocol version as implemented
4	Byte	Release within version as implemented
5 ... 7	Bytes	don't care

13 Error Handling

Calibration session state diagram and error handling procedures

The state diagram shows the state transitions and the possible error categories. The table below shows the action taken after these error conditions.



'Cold Start' incorporates:
- CONNECT to establish a logical link inbetween master and slave device,
- *seed&key* log-in procedure,
- Automatic session configuration (using EXCHANGE_ID) unless overrided,
- Automatic code update, if supported by the master.

Within the state 'Calibration data initialization', 'DAQ list initialization' or 'Session running' the CCP master device may request to repeat embedded *seed&key* procedures.

Category	Description	Action	Retries
timeout	no handshake message	retry	2
C0	Warning	-	-
C1	spurious (comm error, busy, ...)	wait (ACK or timeout)	2
C2	resolvable (temp. power loss, ...)	reinitialize	1
C3	unresolvable (setup, overload, ...)	terminate	-

The error category C1 is handled by a retry and does not lead into a state change.

14 Example Sequences

For some commonly implemented services the procedures using CCP are explained. This only shall be considered as an example on the useage of the CCP. The settings of the session status byte are described in the following form:

(01xx xx1x) means: x ⇨ bit remains unchanged; 0/1 ⇨ bit is forced 0/1

Macro service sequences are terminated if unresolvable errors (Category 3) occur.

14.1 Session log-in

Command	Description	Remark
CONNECT	Connect logically	
GET_CCP_VERSION	Aggree on protocol version	
EXCHANGE_ID	Exchange station identifications	Plug 'n Play
GET_SEED	Get seed for key, returns key	if 'seed&key' is used
UNLOCK	Unlock protection by sending received key	if 'seed&key' is used
SET_S_STATUS	Set session status (set one or more status bits)	

14.2 Block DownLoad

Command	Description	Remark
CONNECT	Connect	Bypass, if already connected
SET_MTA	Set memory transfer address to destination block	
DOWNLOAD	n • data download, corresponding to size of the block to be downloaded	

14.3 Block UpLoad

Command	Description	Remark
CONNECT	Connect	Bypass, if already connected
SET_MTA	Set memory transfer address to source block	
UPLOAD	n • data upload, corresponding to size of the block to be uploaded	

14.4 Calibration Data Initialization

Command	Description	Remark
CONNECT	Connect	Bypass, if already connected
SET_S_STATUS	Set session status (xxxx xxx0)	CAL = off
loop	n •	
SET_MTA	Set memory transfer address to destination block	
BUILD_CHKSUM	Build checksum of block	
DOWNLOAD	download block, if checksum does not match	
SELECT_CAL_PAGE	Select calibration data page	
SET_S_STATUS	Set session status with bit CAL=1 (xxxx xxx1)	now calibration session is running

14.5 DAQ List Initialization

Command	Description	Remark
CONNECT	Connect	Bypass, if already connected
SET_S_STATUS	Set session status with DAQ=0 (xxxx xx0x)	
GET_DAQ_SIZE	Allocate DAQ list	see section 'Organization of DAQ'
loop	n •	
SET_DAQ_PTR	Set destination pointer	
WRITE_DAQ	write list data	
SET_S_STATUS	Set session status with DAQ=1 (xxxx xx1x)	
START_STOP	Start transmission of DAQ DTOs and set parameters	

14.6 Code Update

Command	Description	Remark
CONNECT	Connect	Bypass, if already connected
SET_MTA	Set memory transfer address to start address	
CLEAR_MEMORY	clear memory of slave device	
loop	n •	
PROGRAM	PROGRAM depending on size of sector or device	
PROGRAM	Size = 0	End of programming sequence

This procedure may also be embedded in a service used for programming of a FLASH EPROM.

15 Expected Performance Ratings

Performance ratings of this protocol are strongly depending on the response latency times, the bus baudrate, the bus load situation and the bus priority level of the protocol objects.
As only small portions of memory may be transferred at a time, the latency times of the involved network nodes is the most restrictive factor regarding burst transfers.

With a baudrate of 500 kbit/s and typical load conditions for the bus and for the nodes (ECU´s) the following ratings have been achieved:

Type	Data rate (approx.)	Remark
burst memory transfer	4-6 Kbyte/s	
async. data acquisition	20 Kbyte/s	overall sampling rate

16 Appendices

In this Appendix additional information on the use and implementation of the CCP is given.

16.1 Matrix of Error Codes

CCP Command	CCP Error Code			
	Access denied 0x33	**Parameters out of range 0x32**	**Access locked 0x35**	**DAQ list init request 0x22**
CONNECT	Session currently not possible	Invalid station address		
EXCHANGE_ID	Illegal master ID			
UNLOCK			Unqualified key	
SET_MTA	Attempted read of classified data	Illegal MTA#, base address, address ext.		
DNLOAD	Write attempt to ROM	Data block size >5		
DNLOAD_6	Write attempt to ROM	Received data block < 6 bytes		
UPLOAD	Upload of classified data	Data block size >5		
SHORT_UP	Upload of classified data	Block size > 5, illegal base addr./addr. ext.		
SELECT_CAL_PAGE	currently not possible			
GET_DAQ_SIZE	No DAQ privilege level	Illegal DAQ list number, wrong CAN ID		
SET_DAQ_PTR	No DAQ privilege level	Unknown DAQ list, illegal ODT# /		
WRITE_DAQ	No DAQ privilege level	Illegal size of ODT element/addr./addr. ext.		
START_STOP	Start/stop of nonexistent DAQ list	Illegal last ODT/line transm. interval, illegal start-stop param.		Starting DAQ of an unconfigured DAQ list
DISCONNECT	Temporary disconnect not possible	Invalid station address, invalid temporary disconnect param.		
SET_S_STATUS	status bits violate master privilege level	Currently illegal combination of status bits		
GET_S_STATUS	Session status not accessible			

16.2 Broadcast Techniques in Multi-point Applications

In certain multi-point applications it might be useful to carry out commands in more than one slave device simultaneously. Although the CCP master-slave command protocol has been designed to work point-to-point with only ***one*** slave connected at a time, there are several approaches for broadcasting protocol commands to more than one slave station.

1) Broadcast station address

If all concerned slave stations recognize a common broadcast address (in addition to their dedicated slave address) the CCP master device may multi-***connect*** to those slaves.

2) Nested connect commands to multiple stations without disconnect

In every case, each slave independantly has to transmit ***Command Return Messages*** to every command and the CCP master has to verify the return messages separately.

Broadcasting may offer considerable advantages regarding synchronization and speed. However, error handling during broadcast connections becomes obviously very complex and demanding.