Introduction To CAN Calibration Protocol

Version 2.0
02/09/03

Application Note AN-AMC-1-102

Author(s) Kim Lemon
Restrictions Public Document
Abstract This application note introduces the CAN Calibration Protocol used to calibrate controllers during module development. The paper also discusses how to implement a CCP Driver.

Table of Contents

1.0 Overview

While the CAN Calibration Protocol or CCP is a reasonably well known standard in Europe that continues to gain acceptance, its use in the American automotive electronics arena has to some extent been limited to the engine calibration area. Yet the CCP is not just for calibration. It has many general-purpose features; such as flash

Contact Information: www.vector-cantech.com or 1-248-449-9290

programming capability, which makes it useful for a wide range of module development activities. CCP users have access to online measurement data and can calibrate modules, so software development can occur not only in a lab environment but also during an in-vehicle test.
Even though many U. S. companies are using or evaluating the CAN Calibration Protocol, many product development engineers are unaware of this potentially reusable software. To help raise the level of awareness for this "new to the U.S." technology, this paper introduces the history, purpose, and the upper-level structure of the CAN Calibration Protocol. Finally, there is a brief discussion of a CCP Driver implementation.

1.1 Introduction

The CAN Calibration Protocol, commonly referred to as CCP, is essentially a software interface used to interconnect a development tool with an Electronic Control Unit (ECU). The interface defines methods to handle module calibration, measurement data acquisition, and flash programming activities. The module developer's needs decide whether the complete CCP interface or a portion of it is supported by the tool or implemented in the ECU. Based on the Controller Area Network (CAN) protocol, CCP places no limitations on the choice of physical layer or the system-selected communication bit rate.
CAN Calibration Protocol is capable of supporting a single point-to-point connection or a networked connection to an entire distributed system. This means that any combination of calibration, measurement data acquisition, and flash programming activities are possible for a single module or any number of modules across a CAN network.
CCP was adopted and enhanced by a European task force called ASAP (German for "Working Group For Standardizing Application Systems"), founded by German companies including Audi, BMW, Mercedes-Benz, Porsche, and Volkswagen. The ASAP standards effort has been renamed into a new organization called ASAM (Association for Standardization of Automation and Measuring Systems). The new organization has expanded the scope of the original ASAP interfaces beyond measurement and calibration to include diagnostics.
The purpose of the ASAM task force is to standardize calibration, measurement, and diagnostics systems to provide compatiblity between hardware and software components and the exchangeability of data.
Instead of each OEM creating its own independent interface and tool architecture to support electronic control unit development, a common interest in creating an external support infrastructure for development tools was instrumental in establishing the ASAP task force. Many manufacturers of calibration equipment, test and development systems have joined this effort in addition to automotive OEMs. ECUs with different microcontrollers and various operating systems can all be compatible with a single off-the-shelf calibration and measurement tool.
Based on requirements including compatibility of software, hardware and information exchange, the ASAP task force originally defined three system-level interfaces:

- ASAP1
- ASAP2
- ASAP3

With the change of ASAP into ASAM the interfaces have also been renamed. The new interfaces for ASAP1, ASAP2 and ASAP3 are ASAM-MCD1, ASAM-MCD2, and ASAM-MCD3 respectively. As the final certification process for ASAM-MCD compliance is not complete, this document refers to the original terms in which the tool and the CCP driver that will be referenced are indeed compatible.
Figure 1 shows a general system level diagram with these three interfaces. A Measurement and Calibration System is connected to an Electronic Control Unit (ECU) via the ASAP1 interface. The ASAP1 is split into Interface 1a and Interface 1b. Interface ASAP1a is a physical and logical data link between the ECU and Measurement and Calibration Systems, and the Interface ASAP1b is the integration of interfaces into the Measurement and Calibration system. A suitable data exchange database description is provided by the ASAP2 interface specification. The interface ASAP3 connects the Measurement and Calibration System to an automation system to provide the capability of automatic computer-controlled testing.

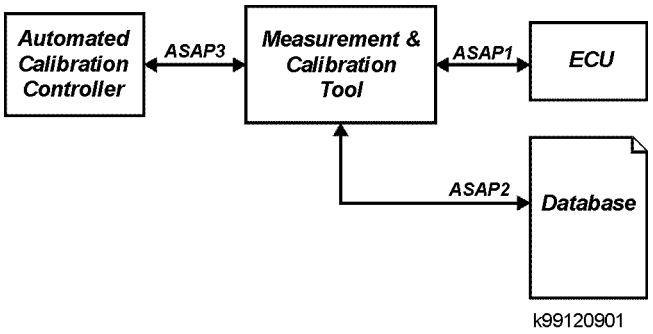


Figure 1: The ASAP Interfaces

While the ASAP1 interface is split into two components, ASAP1a and ASAP1b, it is the ASAP1a component that is directly related to CCP. To some extent independent of protocol, ASAP1a supports data exchange by using either CAN (Controller Area Network) or a UART-based asynchronous serial data link (K-Line or ISO 9141). It is also important to note that although ASAP1a does not currently include KWP2000-type diagnostic services using CCP, ISO Task Force 2 (TF2) is working toward this definition. ASAP published the CAN Calibration Protocol Version 2.0 in June, 1996. The current CAN Calibration Protocol Version 2.1 was released in February 1999.

1.2 CCP Basics

The CAN Calibration Protocol is basically used as a monitor program. Similar to many earlier serial RS232-type monitors and bootstrap loaders that provide basic read and write memory capabilities, CCP provides the same functionality using a standard protocol rather than a company-specific proprietary protocol. However, when a rather high-speed CAN bus is used, CCP, unlike some previous 9600 baud UART-based monitor, provides the ability to access data at such a fast rate that it is possible to run an application at the same time.
Developers now have a significant advantage over the earlier monitor methods.
In the dialog used by CCP and most monitor programs it is the tool or PC that is the master of the commands sent into the ECU. The ECU does nothing without the master (Tool) initiating commands. Using the appropriate CCP messages, a CCP-compliant tool can read data from the ECU and can write data into the ECU.

With CCP, the software developer can read:

- RAM
- PORTS
- ROM
- FLASH

With CCP, the software developer can write to:

- RAM
- PORTS
- FLASH

However, this is only CAN Calibration Protocol's minimum capability. CCP includes several additional monitor commands, and provides several new features including automatic data acquisition processing based on events or periodic updating, flash programming and data security. Since there is no requirement to use all its features, CCP is a scalable protocol.

2.0 CCP Dialog

CCP uses a specific conversation or dialog to accomplish each designated function with only two CAN identifiers for message transfers. Each dialog is a collection of exchanged messages between a master, the calibration or development tool, and a slave ECU.
Most CCP dialog always uses a master/slave form of conversation. The tool (or master) always initiates the conversation with a single CAN message, and once this message has been received, the ECU (or slave) is then responsible for responding with a single CAN message.

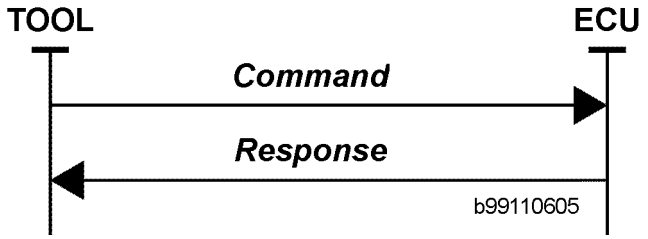


Figure 2: General CCP Dialog

One of the CAN identifiers is used to send information from the tool to the ECU. This command is defined from the ECU's point of view as the Command Receive Object (CRO). The CRO contains command information and the related parameters needed for the command. After the ECU receives the CRO, the CCP driver within the ECU processes the command code in the CRO. Then internal functions or data transfer between the tool and the ECU can occur.

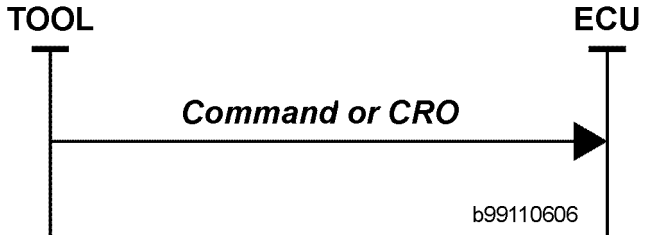


Figure 3: Command Receive Object

2.1 CCP Commands

The following table shows an overview of all CCP commands used in the CRO:

Command	Description
CONNECT	Establish a logical connection to a slave device
EXCHANGE_ID	Get the ECU identification
TEST	Test for presence of a slave device
SET_MTA	Set memory transfer address (MTA)
DNLOAD	Download up to 5 bytes into the ECU memory.
DNLOAD_6	Download 6 bytes into the ECU memory
UPLOAD	Upload up to 5 bytes from ECU memory

SHORT_UP	Upload up to 5 bytes from ECU memory (no MTA)
GET_DAQ_SIZE	Get the size of a DAQ list
SET_DAQ_PTR	Set the ODT entry pointer
WRITE_DAQ	Write an entry in a ODT
START_STOP_ALL	Start or stop all DAQ lists
START_STOP	Start or stop a specific DAQ list
DISCONNECT	Finish a logical connection
SET_S_STATUS	Set the ECU session status (optional)
GET_S_STATUS	Get the ECU session status (optional)
BUILD_CHKSUM	Calculate a memory checksum (optional)
CLEAR_MEMORY	Clear a memory range (optional)
PROGRAM	Download up to 5 program bytes (optional)
PROGRAM_6	Download 6 program bytes (optional)
MOVE	Move a memory range (optional)
GET_ACTIVE_CAL_PAGE	Get the active calibration page (optional)
SELECT_CAL_PAGE	Select the active calibration page (optional)
UNLOCK	Unlock the access protection (optional)
GET_SEED	Get the access protection code (optional)

These commands are all represented by hexadecimal numbers in the first byte of the CRO message.

2.2 CRO Message

The data field for each CAN Identifier used with CCP is limited to 8 data bytes because CCP is based on CAN. The CRO uses each of the data bytes for specific items of information based on the command.

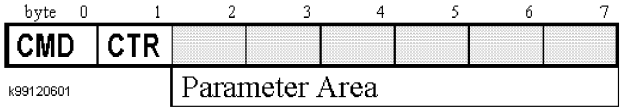


Figure 4: Structure of the CRO Message

The first byte of the CRO Message is a command number that corresponds to the CCP command as defined in the specification. The second byte is a command number counter from the tool to track the current command that was issued. This same value will also be used in the response from the ECU to the tool.

2.2.1 Basic CCP Response

The other CAN Identifier is used to send information from the ECU to the tool. This command, also defined from the ECU's point of view, is the Data Transmission Object (DTO).

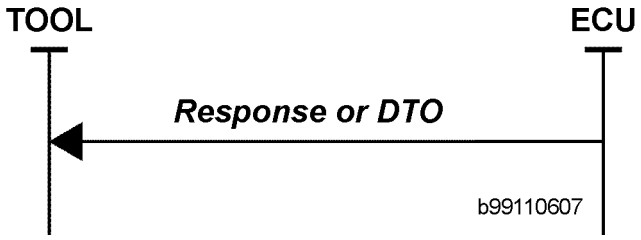


Figure 5: Data Transmission Object

Three types of DTOs are defined by the CCP specification. These three types are the Command Return Message (CRM), the Event Message, and the Data Acquisition Message (DAQ).

2.3 CRM-DTO Message

The Command Return Message (CRM) is a message sent by the ECU in response to a CRO. The CRM can be a simple acknowledgement to the CRO (e.g., essentially saying "I'm here" to the CCP commands CONNECT or TEST), or it can contain actual requested data. The CCP specification describes in detail the content of each CRM for each CRO.

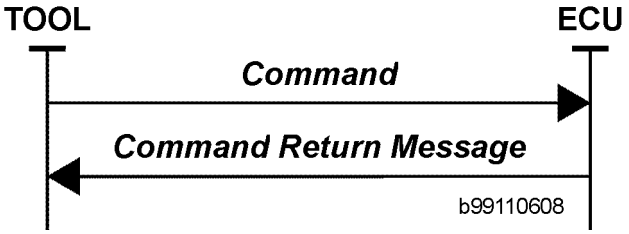


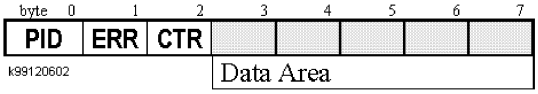
Figure 6: Command Return Message

2.4 Event Message

The Event Message is a specific type of DTO used to inform the tool of ECU internal status changes caused by events. This information can then be used to invoke error recovery or other services. The main purpose of the Event Message is to allow the ECU to report any errors to the tool that have occurred since the last CRO was sent by the tool. A list in the CCP specification describes the available error codes.

Figure 7: Event Message

The CRM and the Event Message have the same structure for the first three bytes of the message. The first byte is the Packet Identifier (PID). A CRM has the value 0xFF, and the Event Message has the value 0xFE. The second byte of both is the error code, since a CRM can also return an error code if e.g. it cannot access an address. The third byte is the command counter value sent by the tool in the last CRO message. The remaining bytes are used for data relating to a particular response.



PID: PID=255: Command Return Message.
PID=254: Event Message.
ERR: Error code.
CTR: Command counter as received in CRO with the last command.

Figure 8: Structure of CRM and Event Message

2.5 DAQ-DTO Message

The Data Acquisition (DAQ) message is a specific type of DTO used to send measurement data to the tool. First, though, the ECU must be initialized. The tool must send initialization messages telling the ECU which measurement data needs to be sent to the tool. The tool also sends information on whether the measurement data is event driven or periodically sampled. Then the measurement data values can be sent to the tool without the tool first having to request the information in a CRO.

Figure 9: Data Acquisition Message

Once initialization is finished, the ECU automatically sends the measurement data at the appropriate time. The data required for a particular event or time period is stored at that time. The stored data can then be transmitted in a DAQ message. This ensures that the measurement data is synchronous to the particular event or time period.

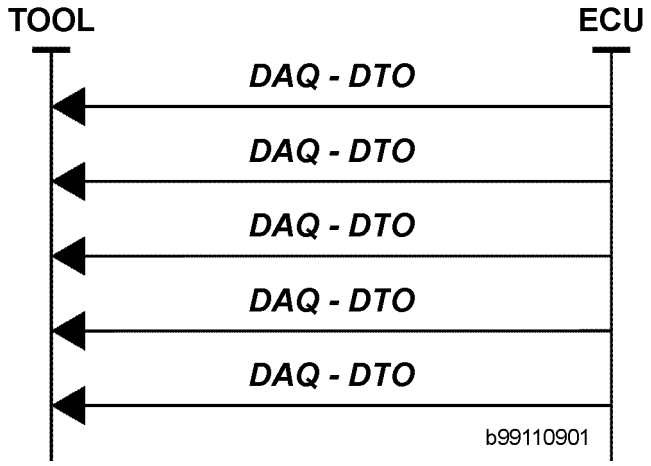
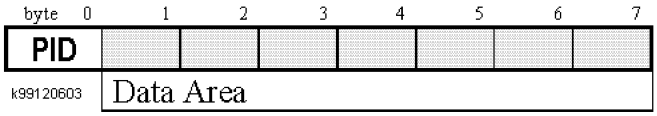


Figure 10: Synchronous Data Acquisition

Tables are set up in the ECU during initialization to identify the location of the measurement data (see Fig. 12). Each table can contain up to seven memory addresses. The table is then assigned a unique packet identifier PID that is placed in the first byte of the DAQ message. The other seven bytes send the requested data to the tool.



PID: Packet Id
PID=n: DTO contains data corresponding to an Object Descriptor Table.

Figure 11: Structure of Data Acquisition Message

2.6 Object Descriptor Tables

The tables that are used to organize the location of the measurement data are called Object Descriptor Tables (ODTs). An ODT describes the contents of one CAN message for data acquisition. Each ODT stores up to seven address locations where the measurement data is stored. The unique Packet Identifier PID is then assigned to the ODT to identify the data. Since a unique PID is needed to identify the measurement data for each ODT, and the number of data bytes that is sent back per ODT is limited to seven data bytes, multiple ODTs may be needed to store all the requested measurement data. A DAQ list is a list of all the ODTs for a particular event or time period. Multiple DAQ Lists are needed when data needs to be acquired based on different events or time periods.

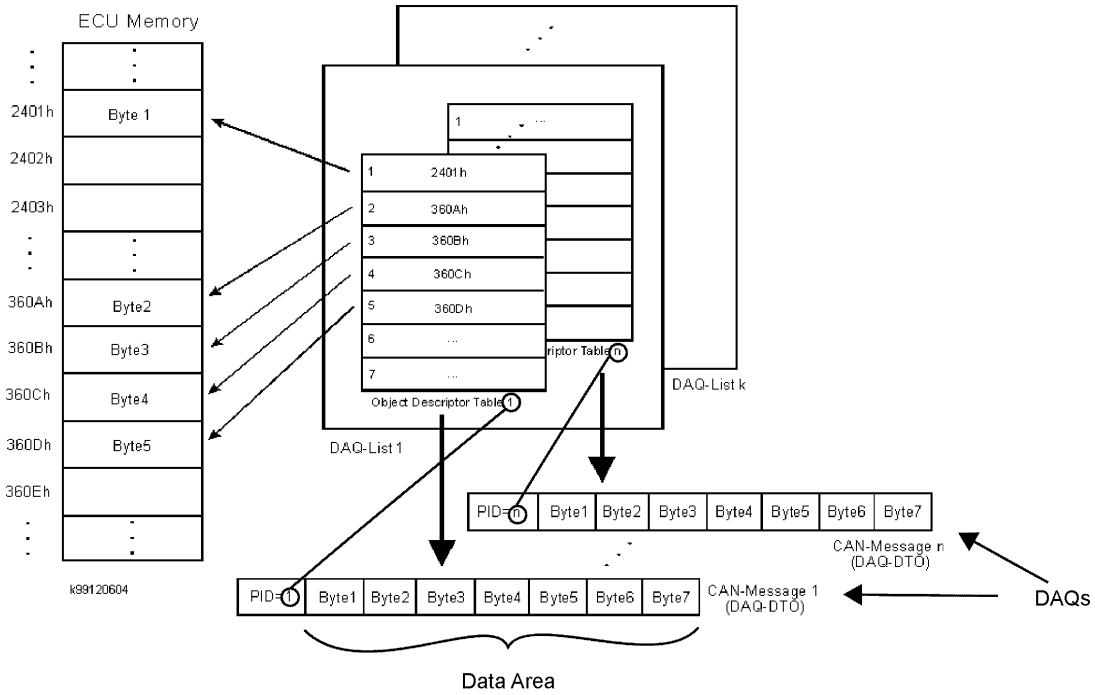
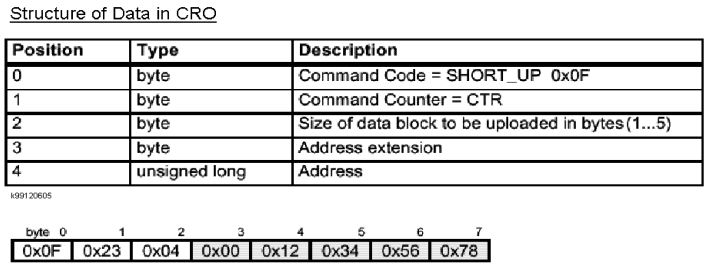


Figure 12: Overview of DAQ Lists with Multiple ODTs

2.7 Example Of A CCP Command: SHORT_UP

A good example of how all this works is SHORT_UP, a command defined in the CCP specification that is used to request a data block of a specified size from the ECU starting at a particular address. The tool will send the size of the data block and the starting address in the CRO command. The command code and the command counter are sent in the first and second byte respectively. Figure 13 shows the data needed for the SHORT_UP command. The first data byte (byte 0) is the corresponding command code for SHORT_UP, which is 0x0F. The second byte (byte 1) is the current value of the command counter. The third byte (byte 2), is the size of data to be uploaded. This value can range from 1 to 5 (bytes of data). The fourth byte (byte 3) is the address extension if applicable for the implementation. The last four bytes contain the address of the data to be uploaded to the tool.



Structure of Data in CRO

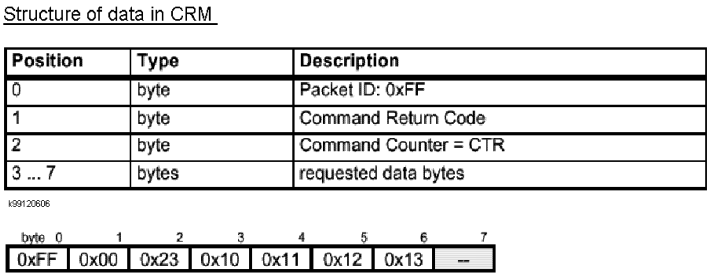
Position	Type	Description
0	byte	Command Code = SHORT_UP 0x0F
1	byte	Command Counter = CTR
2	byte	Size of data block to be uploaded in bytes (1...5)
3	byte	Address extension
4	unsigned long	Address

k99120605

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

Figure 13: Example: SHORT_UP CRO

The ECU receives the CAN message with the SHORT_UP command and responds to this message with the appropriate CRM. Figure 14 shows the data needed for the SHORT_UP response. The first byte (byte 0) of the response is a Packet Id of 0xFF. This indicates that the DTO is a CRM. If the ECU responds successfully to the request, the second byte (byte 1) is the command return code with the value 0x00. (The complete list of command return codes is listed in the CCP specification.) The third byte (byte 2) is the command counter value received in the CRO. The last five bytes contain the actual requested data. Since in this example only four data bytes were requested by the CRO, the tool disregards the data in the last byte (byte 7).



Structure of data in CRM

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

k99120606

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

Figure 14: Example SHORT_UP CRM-DTO

The CCP driver in the ECU is developed so that it will support the commands described in the CCP specification. The commands must be received from the CAN bus and processed so that the ECU can provide the appropriate response, which must be sent back out on the CAN bus. There are two possible ways to obtain measurement data: 1) A simple polling method can be implemented that sends data only after a request message from the tool, or 2) DAQ lists can be implemented when more throughput of data is needed or when the data needs to be obtained synchronously.

3.0 CCP Communication

The CCP driver can be implemented with all the commands in the CCP specification or scaled down to include only those commands needed for a particular implementation. The CCP implementation in the ECU consists of two parts. The first part is a command processor, which allows the ECU to receive the required CRO commands and send the appropriate CRM. The second part involves a DAQ processor, which is responsible for sending the required DAQ list information at the appropriate time. Figure 15 illustrates the two main components of the CCP Driver.

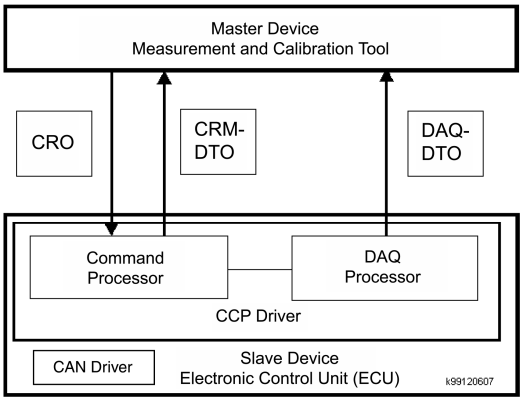


Figure 15: CCP Communication

In summary, the ECU receives commands codes and the related parameters in the Command Receive Object (CRO) to carry out internal functions or memory transfers. The Data Transmission Object (DTO) is used by the ECU to respond to the CRO, to indicate any error conditions, and to transmit measurement data to the tool.

3.1 Tool Considerations

When CCP is used with a tool like Vector's CANape, which is capable of displaying ECU data in physical units, in symbolic value (i.e. ON, OFF, ENABLED, DISABLED), or in graphical form, the module developer has many new capabilities beyond the earlier tools.

3.2 CCP Applications

Basic development uses for CCP include:

- Real-time information from the ECU (basic read and write functions)
- Real-time access to ECU parameters (data acquisition)
- Real-time adjustment of the ECU's process algorithms (Calibration)
- In-system or in-vehicle evaluation of design concepts
- Evaluation of engineering design modifications
- In-system (or in-vehicle) Flash Programming
- Emulation-type operation beyond the lab bench

Coupled with the right tools, CAN Calibration Protocol is suitable for several module development activities.
CCP allows development outside of the traditional software engineering environment. Beyond the engineer's desk and the engineering lab, module development on the road or on the test track is now possible, and several companies are already at this advanced stage.

4.0 CCP Driver Implementation

Before any interaction with the tool can occur the CCP Driver must be implemented in the ECU. The tool sends the CCP commands in a particular order to get the appropriate information from the ECU. The ECU only needs to implement the correct responses to each command as required by the CCP specification.
The CONNECT command must be processed like other CCP commands; however, the ECU must also store additional information about the status of the connection. As required in the specification, the ECU must not respond to any CCP messages unless it has first processed a CONNECT command with the proper station address. Therefore, the CCP driver must store the current status of the CCP connection so that commands can be ignored or acknowledged as required by the CCP specification.

5.0 Initialization Sequence

Initialization in a typical tool application proceeds as follows. The tool sends the CONNECT command with the station address of the ECU. If the station address is correct, the ECU responds with the correct CRM-DTO. Next, an option would be to issue the GET_CCP_VERSION command to allow the tool to determine if the ECU is compatible with CCP implementation 2.0 or 2.1. Then, the EXCHANGE_ID command can be issued for automatic session configuration. The ECU responds by sending the Station Identification name's length and setting the Memory Transfer address to the appropriate memory location with the Station Identification name. The tool can then request the Station Identification name.
A measurement and calibration tool may use an ASAP2 -compliant database containing the descriptions of all ECU objects. Each version of the ECU software can be described by a different database. The EXCHANGE_ID can be used to make sure the tool database version matches the ECU software version. Figure 16 illustrates this sequence.

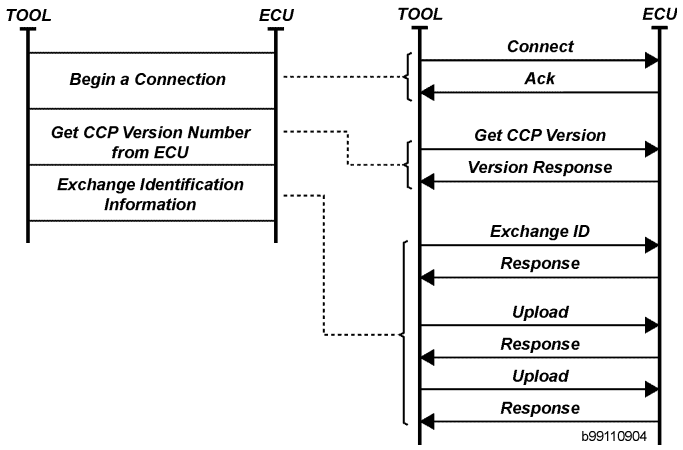


Figure 16: Initialization Operations (Example)

If optional features such as Seed & Key are implemented, the tool can send commands for security access. The next step is to synchronize the calibration values currently in the ECU by comparing them with the calibration values that the tool needs for display. There is also a checksum feature that can be implemented. If both the ECU and the tool create a checksum on the needed calibration values and the value is the same, then the synchronization is complete. Otherwise, the tool may need to upload or download the parameters based on user request.

6.0 DAQ Operations

If DAQ lists are used for data acquisition in a typical tool application, the tool must first inquire about the DAQ storage information in the ECU. For example, the user may request information based on two DAQ lists but find that the ECU

has only been configured to store information for one DAQ list. After the tool determines that the setup in the ECU is sufficient for the current requested measurement data, then the DAQ lists can be configured and the command to start DAQ information transfer can be sent by the tool. The ECU must continue to send the required measurement data until the DAQ STOP command is received from the tool. Figure 17 illustrates an example of the DAQ operations sequence.

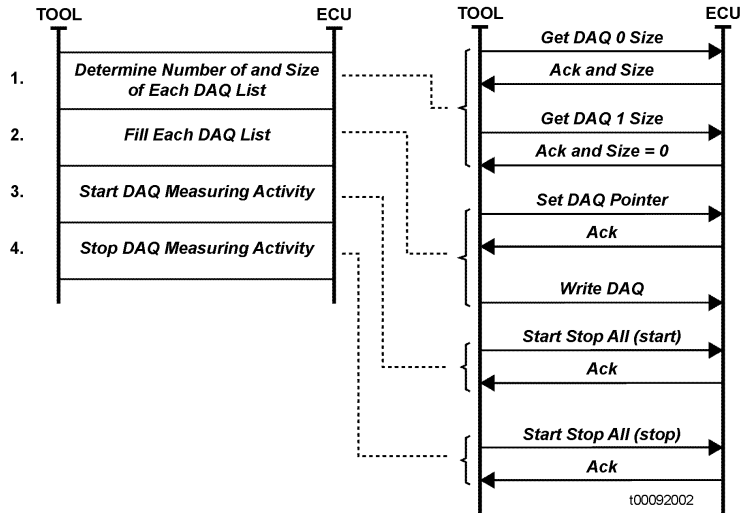


Figure 17: DAQ Operations (Example)

The ECU keeps sending the size information to the tool. When it finally sends Size = 0, this means, "I have no more DAQs." The tool then sets the DAQ pointer to fill each DAQ list and writes the DAQ.

Disconnect
When the tool and ECU no longer need to be connected, the tool can issue the DISCONNECT command. The CCP specification allows either a temporary disconnect or a complete termination of the calibration session.

7.0 CCP Implementation Requirements

The software developer needs three items to implement CCP:

- CCP specification document
- CAN bus connection to the ECU
- CCP Software Driver

The first item, the CCP specification document, can be located on the ASAM web site at the following link:

http://www.asam.de/Asap_Gen.htm

This link also has a number of other public documents produced by ASAM.

Secondly, the CAN bus connection to the ECU must be developed by the ECU supplier. CAN software drivers are also available from various sources, or the ECU supplier can create custom CAN software drivers.

The third item, the CCP software driver, must be created. This software can be a custom implementation developed by the user, or a free CCP driver can be downloaded from the ASAM link. The CCP driver was developed by Vector Informatik. The ccp.zip file contains the CCP driver, a sample PC-based simulator implementation, and supporting documentation, which contains one document with details on integrating the CCP driver into an application.

7.1 CCP Driver Implemention Example

The free CCP Software driver implementation interfaces to the application and the CAN driver. The CAN driver must be able to receive CAN messages and send the information in the CCP CAN message to the CCP driver. All other CAN messages can then be received and processed by the application for normal operation. Figure 18 illustrates the functionality needed for CCP CAN message reception.

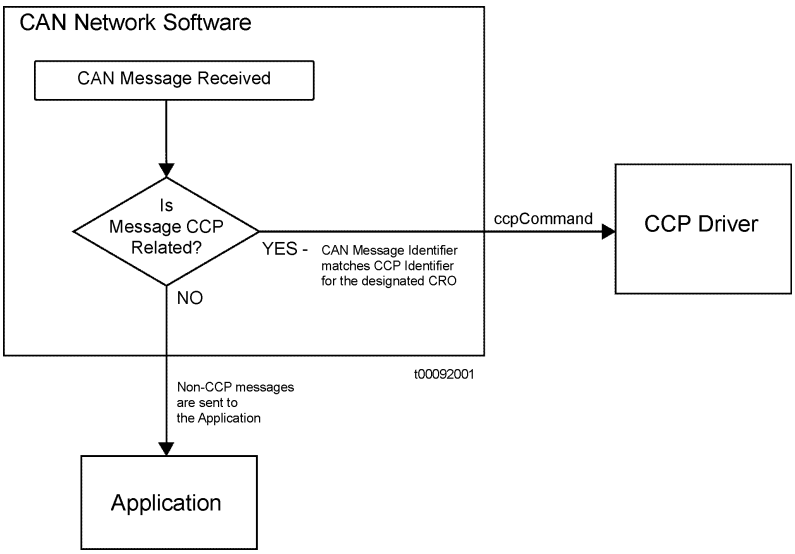


Figure 18: CCP CAN Message Reception

If the identifier of the CAN message received matches the CCP identifier for the Command Receive Object (CRO), the message is CCP related, and is sent to the Driver. If the identifier of the CAN message received does not match the CCP identifier, the message is not a CCP message, and so is sent on to the application.

In order for CCP messages to be transmitted, the CCP driver code must also be able to interface to the CAN driver code. When the CCP driver calls a function in the CAN driver to transmit the CCP message, the CCP driver needs to be informed when the message has been transmitted successfully. When the CAN driver calls a function (ccpSendCallBack) to inform the CCP driver that the last CCP message was transmitted successfully, the CCP driver can then call the CAN driver function to send another message. This confirmation process is useful in two ways: 1) It prevents the CCP Driver from overloading the CAN transmit buffer with CCP messages, and 2) if reception and transmission are integrated into the application's operating system, it gives the application some control over how often the CCP messages are sent. Figure 19 illustrates the functionality needed for CCP CAN message transmission.

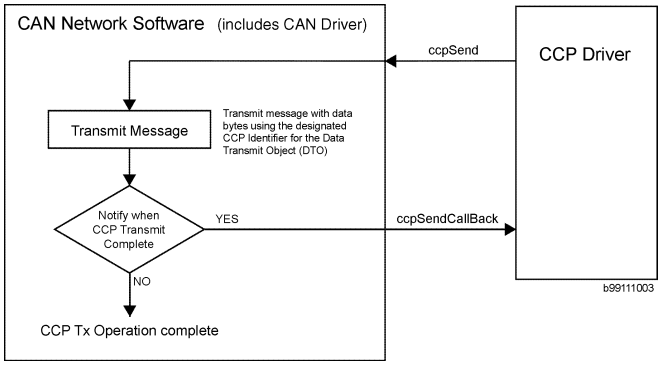


Figure 19: CCP CAN Message Transmission

The sample CCP driver must also be interfaced directly to the application. An initialization routine (ccpInit) must be called when the application starts. If synchronous data acquisition is implemented, then the application must call a function to process the measurement data (ccpDAQ). If a checksum is implemented for calibration data, then the function 'ccpBackground' must be called to calculate the checksum. A function for distinguishing different address spaces or memory types can be used if necessary. There is also an optional function for providing write protection for certain memory addresses. Figure 20 gives an overview of the interfaces needed to implement the sample CCP driver. For more information on how to implement Vector's free CCP driver, please refer to the CCP Driver Implementation in Electronic Controls Units in the Reference section.

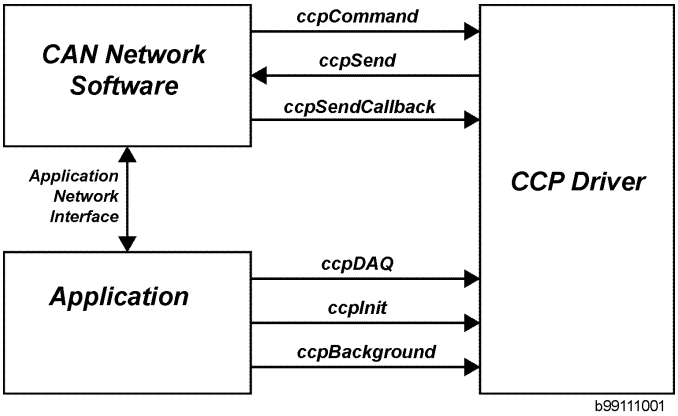


Figure 20: CCP Sample Driver Interface

7.2 CCP Resource Requirements

The CCP software driver uses resources such as RAM, ROM and CPU time in the ECU. The code size of the CCP software driver depends on which optional features are implemented.

CCP Resource Example

The following list shows the resource requirements for an implementation with 1 DAQ List and 3 ODTs. This allows for storing of up to 21 bytes of data in an intermediate buffer.

- RAM
  83 bytes (21 bytes for cyclic measurement data acquisition, 46 bytes for the DAQ lists, and 16 bytes qfor the transmit and receive buffers)
- ROM
  1-2 kBytes (depending on which CCP options are implemented)
- CPU time
  CCP Execution Times (80C167 / 16 MHz)
  100 – 200 µSec (depending on command)
- Creating and sending a CAN message per ODT
  50 – 100 µSec (depending on bus load)

7.3 CCP Performance Ratings

CCP performance depends on a number of factors. The response latency time of the services in the ECU affects CCP performance: the amount of time allocated by the ECU operating system for the CCP driver to perform its functions greatly affects the performance of the CCP driver. CAN bus conditions such as bit rate, busload, and bus priority level of the CCP message also affect the performance of the CCP driver.

An implementation in a Siemens 80C176 16 MHz processor achieved a burst memory transfer of ~5-10 kBytes/s and data acquisition rates of ~25 kBytes/s when the bit rate was 500 kBit/sec and under typical load conditions. A burst memory transfer would include uploading calibration values and flash programming. The data acquisition rate was for synchronous data acquisition of 100 values every 10 msec. Each of the 100 values was two bytes in length.

8.0 Conclusion

Aside from the business advantage of using a standard protocol rather than using a company-specific proprietary solution, CCP provides a wide range of functionality to help both the OEM and the module supplier in the development of electronic modules.

A complete set of tools to handle module calibration, test, measurement, diagnostic, and flash programming activities, all within one protocol, is a big technical advantage for the software engineer. While inventing your own CCP-equivalent might sound attractive, the development community might consider the value of using this existing "off-the-shelf" technology.

9.0 Contact

For more information or to contact the authors, please email info@vector-cantech.com

10.0 References

- CAN 2.0B Specification – 1991, Robert Bosch GmbH
- CP CAN Calibration Protocol, ASAP Standard, Version 2.1, February 1999
- CCP Driver Implementation in Electronic Control Units, Version 1.18, Vector Informatik GmbH, 1999
- CCP, A CAN Protocol for Calibration and Measurement Data Acquisition, Rainer Zaiser, Vector Informatik GmbH

11.0 Additional Sources

The Bosch CAN 2.0B Specification is available at the following website:
http://www.vector-cantech.com

To contact ASAM, visit the ASAM web site at:
http://www.asam.de.

The CCP CAN Calibration Protocol V2.1 and the free CCP driver can be found using the link:
http://www.asam.de/Asap_Gen.htm

12.0 Definitions, Acronyms, Abbreviations

ASAP Arbeitskreis zur Standardisierung von Applikationsystemen (Standardization of Application/Calibration Systems)
ASAM Association for Standardization of Automation and Measuring Systems
CAN Controller Area Network
CCP CAN Calibration Protocol
ECU Electronic Control Unit
UART Universal Asynchronous Receiver Transmitter

13.0 Contacts

Vector Informatik GmbH
Ingersheimer Straße 24
70499 Stuttgart
Germany
Tel.: +49 711-80670-0
Fax: +49 711-80670-111
Email: info@vector-informatik.de

Vector CANtech, Inc.
39500 Orchard Hill Pl., Ste 550
Novi, MI 48375
Tel: (248) 449-9290
Fax: (248) 449-9704
Email: info@vector-cantech.com

VecScan AB
Fabriksgatan 7
412 50 Göteborg
Sweden
Tel: +46 (0)31 83 40 80
Fax: +46 (0)31 83 40 99
Email: info@vecscan.com

Vector France SAS
168 Boulevard Camélinat
92240 Malakoff
France
Tel: +33 (0)1 42 31 40 00
Fax: +33 (0)1 42 31 40 09
Email: information@vector-france.fr

Vector Japan Co. Ltd.
Nishikawa Bld. 2F, 3-3-9 Nihonbashi
Chuo-ku Tokyo 103-0027
Japan
Tel: +81(0)3-3516-7850
Fax: +81-(0)3-3516-7855
Email: info@vector-japan.co.jp