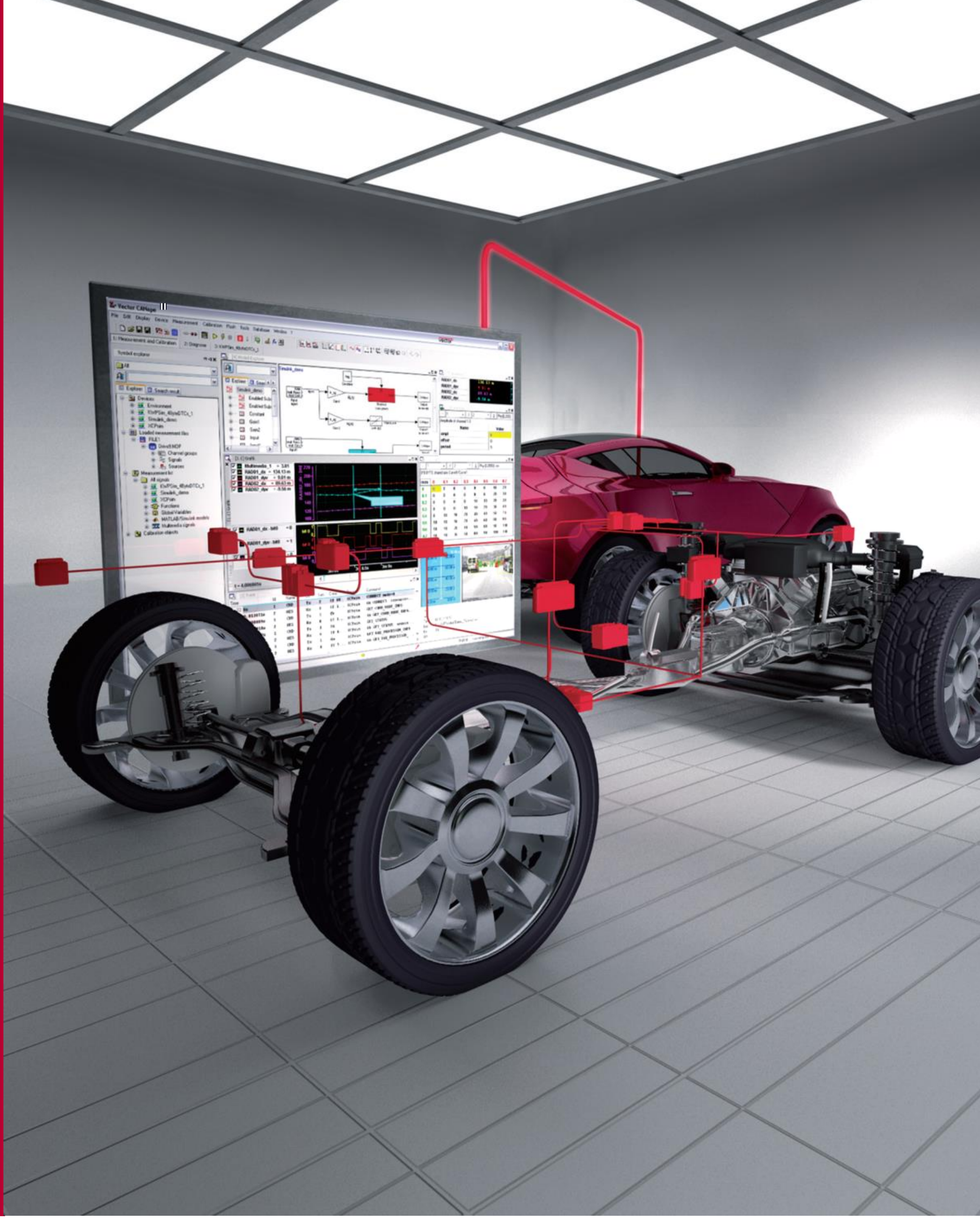


User Manual

AUTOSAR Calibration

Measuring and Calibrating of AUTOSAR
Applications with XCP and CANape

Version 1.0

English

Imprint

Vector Informatik GmbH
Ingersheimer Straße 24
D-70499 Stuttgart

Contents

1 Introduction

In this chapter you will find the following information:

1.1 Purpose of the AUTOSAR Calibration User Manual

AUTOSAR Standard

The AUTOSAR Standard describes methods that enable standardized development of reusable and replaceable software components within vehicles. This approach minimizes the development effort for electronic control unit (ECU) software. The software is then optimized using CANape.

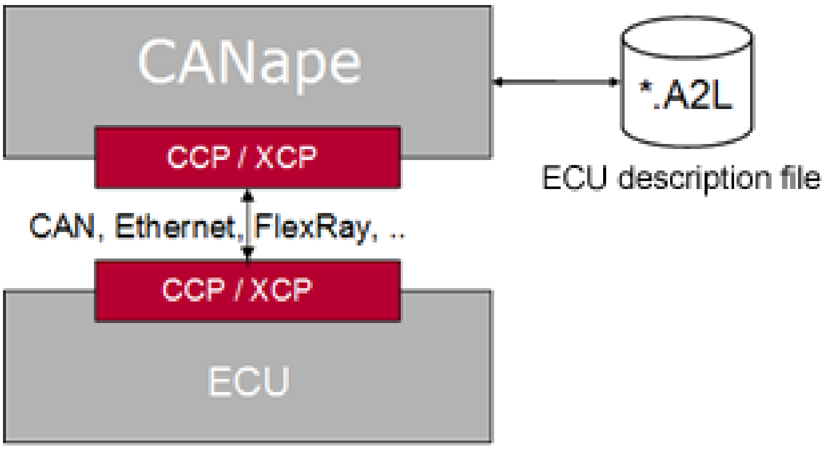
Calibration and measurement parameters

Since the software developer cannot yet optimize the parameters for a control algorithm of the ECU at the time of implementation, these parameters are defined in the software as calibration parameters. The calibration parameters are ultimately variables in the source code that reside in RAM memory and remain unchanged by the algorithm itself. They can then be calibrated using CANape. To record the effects of the calibration process, additional measurement parameters are defined in the software. These parameters are also variables in the source code and reside in RAM memory. In contrast to calibration parameters, however, measurement parameters are continually changed by the ECU algorithm and reflect the current value. This makes the effects of the calibration process visible and allows the behavior of the ECU to be optimized. For example, the wheel speed (calibration parameter) of a driving dynamics control system is changed and the measuring equipment measures the corresponding sensor values (measurement parameters) in order to acquire the change in behavior of the algorithm.

CCP/XCP protocols with A2L file

In order to access the ECU-internal measurement and calibration parameters during runtime, the CCP and XCP protocols are used. A fundamental component of these address-oriented protocols is an A2L file. This file facilitates data handling, since it enables the symbolic selection of data objects independent from their memory addresses in the ECU. Thus, it is possible to access ECU-internal parameters using symbolic names. The measurement, calibration, and diagnostics system (CANape) maintains the link between the ECU-internal addresses and the associated symbolic names. For this, a separate A2L file is required for each ECU. Figure 1-1 shows the integration of the A2L file in the MCD system.

Figure 1-1: Integration of the A2L file in the MCD system



ECU-independent concept

An ECU-independent concept for measuring and calibrating AUTOSAR applications is needed for the development of ECUs based on the AUTOSAR Standard. The AUTOSAR Calibration user manual describes a standardized procedure for implementing and calibrating an ECU according to AUTOSAR.

Structure of this document

The document begins with a brief introduction of the AUTOSAR Standard. Aspects of Measuring and Calibrating of ECU Software are then explained.

The OEM chapter serves as a checklist for OEMs when creating performance specifications. It briefly explains the details that must be communicated to the supplier in order to realize the desired measurement task.

The Supplier chapter describes the procedure on the part of the supplier. It describes details for configuring MICROSAR XCP and the software components of AUTOSAR. It also explains the process of generating the A2L file.

The Delivery Test/Quick Start chapter then explains how CANape can be used to perform a simple delivery test of the A2L file. This can additionally be used as a CANape Quick Start for the OEM.

The final CANape Introduction chapter describes the path from project creation to flashing of optimized parameters in CANape.

1.2 About This User Manual

To Find information quickly

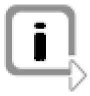
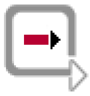
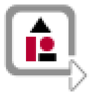
This user manual provides you with the following access help:

> At the beginning of each chapter you will find a summary of the contents.
> The header shows in which chapter of the manual you are.
> The footer shows the version of the manual.
> At the end of the user manual you will find a list of abbreviations to look-up used abbreviations.

Conventions

In the two tables below you will find the notation and icon conventions used throughout the manual.

Style	Utilization
bold	Fields/blocks, user/surface interface elements, window- and dialog names of the software, special emphasis of terms. **[OK]** Push buttons in square brackets **File**|**Save** Notation for menus and menu entries
MICROSAR	Legally protected proper names and marginal notes.
`Source Code`	File and directory names, source code, class and object names, object attributes and values
Hyperlink	Hyperlinks and references.
<Ctrl>+<S>	Notation for shortcuts.

Symbol	Utilization
	This icon indicates notes and tips that facilitate your work.
	This icon warns of dangers that could lead to damage.
	This icon indicates more detailed information.
	This icon indicates examples.
	This icon indicates step-by-step instructions.

1.2.1 Certification

Quality management system

Vector Informatik GmbH has ISO 9001:2008 certification. The ISO standard is a globally recognized standard.

1.2.2 Warranty

Restriction of warranty

We reserve the right to modify the contents of the documentation or the software without notice. Vector disclaims all liabilities for the completeness or correctness of the contents and for damages which may result from the use of this documentation.

1.2.3 Support

Need support?

You can get through to our hotline by calling

+49 (0)711 80670-200

or you can send a problem report to canape-support@vector.com.

1.2.4 Trademarks

Protected trademarks

All brand names in this documentation are either registered or non-registered trademarks of their respective owners.

2 Introduction to AUTOSAR

In This Chapter You Will Find the Following Information:

2.1 Background

AUTOSAR

AUTOSAR (**AUT**omotive **O**pen **S**ystem **AR**chitecture) is a working group of automobile manufacturers and suppliers whose objective is to establish a joint industry standard for automotive E/E (electrics/electronics) architectures.

Main objectives

The main objectives of this effort are:

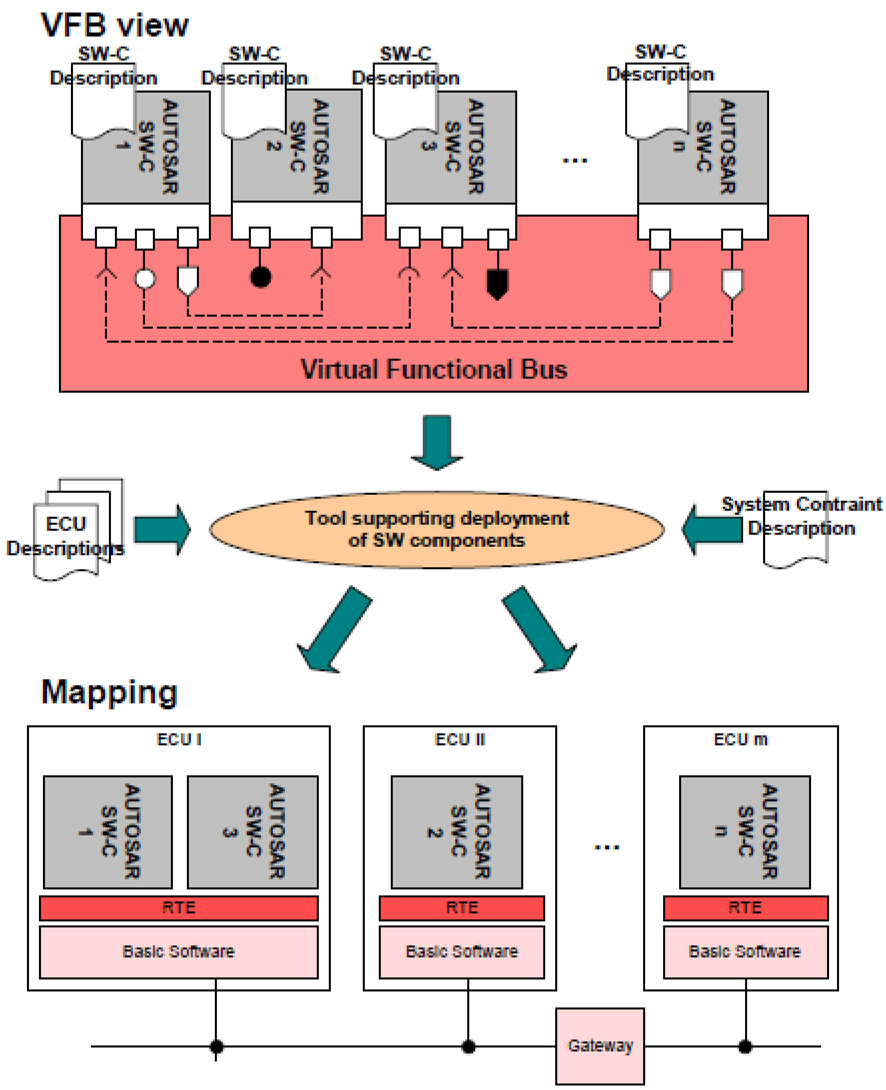
> Management of the increasing E/E complexity
> Improved flexibility for updates and modifications
> Scalability to different vehicle and platform variants
> Improved reliability and quality of E/E systems
> Ability to identify errors in early phases of development
> Reusability of functions irrespective of the supplier
> Standardized model tools and code generators

2.2 Approach

AUTOSAR elements

Figure 2-1 shows the AUTOSAR approach. The individual elements are explained in more detail below.

Figure 2-1: Concept of AUTOSAR[1]



AUTOSAR SW-C

The AUTOSAR software components form the framework of an application that runs on the AUTOSAR infrastructure.

Reference: The interfaces of the AUTOSAR software components are described in more detail on http://www.autosar.org/index.php?p=1&up=2&uup=1&uuup=0.

SW-C Description

The software component description is provided by AUTOSAR, for example, for defining interfaces.

Virtual Functional Bus (VFB)

The VFB describes all communication mechanisms of AUTOSAR at an abstract level.

[1] Source of figure: AUTOSAR Technical Overview V2.2.2 R3.2 Rev 1

System Constraint and ECU Descriptions In order to integrate software components into a network of an ECU, AUTOSAR provides descriptions for entire systems or for configurations and signals of individual ECUs.

Runtime Environment (RTE) The RTE implements the functionality of the VFB of a particular ECU. However, it can delegate a portion to the basic software.

Basic Software (BSW) The basic software provides the infrastructural functionality of the ECU.

2.3 Basic Concept

Communication via VFB

The communication between the individual components takes place via the Virtual Functional Bus (VFB). At this stage, there is not yet any memory management of the ECUs. The VFB is used both within the ECU and across ECUs and has no knowledge of the bus technology used. This enables replacement of the application software, regardless of the bus technology used. Figure 2-2 shows the communication flow of the Virtual Functional Bus.

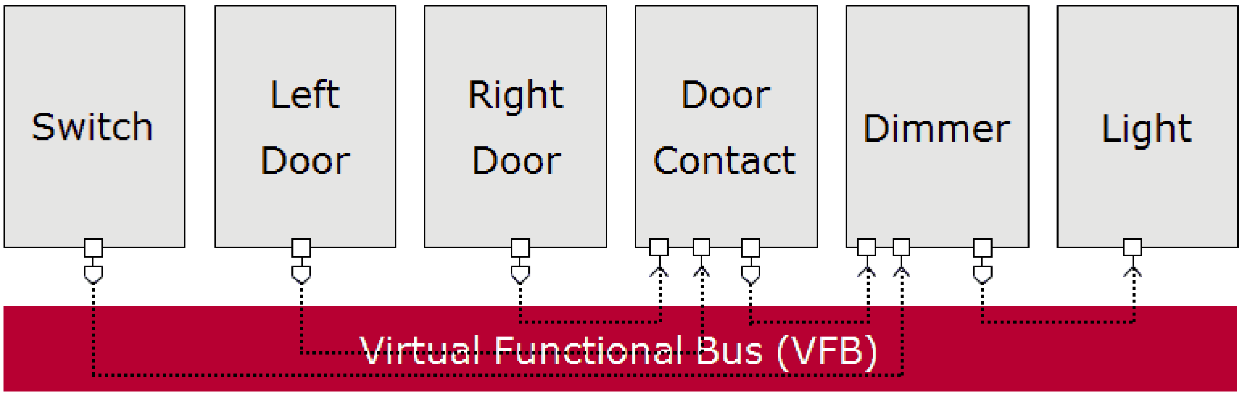


Figure 2-2: Communication flow of the VFB

Running of the components

As soon as all relevant objects have been defined, they are mapped to the ECU. The VFB is implemented using an ECU-specific Runtime Environment (RTE) and, together with the operating system, takes over the running of the components.

Consistence of software components

Software components, here e.g., **Left Door** and **Right Door**, consist of:

> **Ports:** These serve as the interface for communication with other software components. They can act either as sender/receiver or client/server. The ports are interconnected using **connectors**.

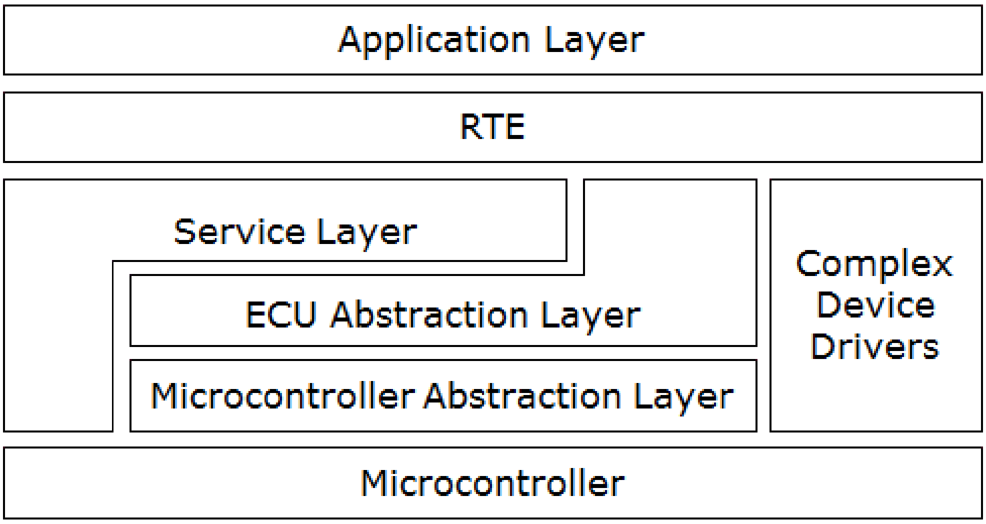
> **Runnables:** Each atomic SW-C contains one or more runnables. These represent the runnable portion of the software component and reference functions and procedures.

2.4 Architecture

Layers

The AUTOSAR architecture essentially has seven different layers (see Figure 2-3). The top and bottom layers are not explained in detail here as they do not belong to the basic software.

Figure 2-3: Overview of AUTOSAR layers



Microcontroller Abstraction Layer

The Microcontroller Abstraction Layer is the lowest software layer of the basic software architecture and provides the upper layers their independence from the actual microcontroller.

ECU Abstraction Layer

The purpose of the ECU Abstraction Layer is to ensure the independence of higher layers from the actual ECU.

Service Layer

The Service Layer is the highest layer of the basic software. It contains the operating system and assumes functions such as the network and NVRAM management and diagnostic services.

Complex Device Drivers

The device driver layer controls special sensors and actuators via direct access to the microcontroller. This involves sensors with special time conditions, for example, that supply fuel injection to paths.

Runtime Environment

As middleware, the Runtime Environment (RTE) integrates different applications with the basic software. It organizes the communication and data exchange between the two layers and manages the running of the runnables. Because all layers are described exactly, the application software can be implemented independent of the hardware and without knowledge of how the other layers behave. The communication between the layers takes place via ports defined beforehand.

The following Figure 2-4 shows the complete AUTOSAR ECU software architecture.

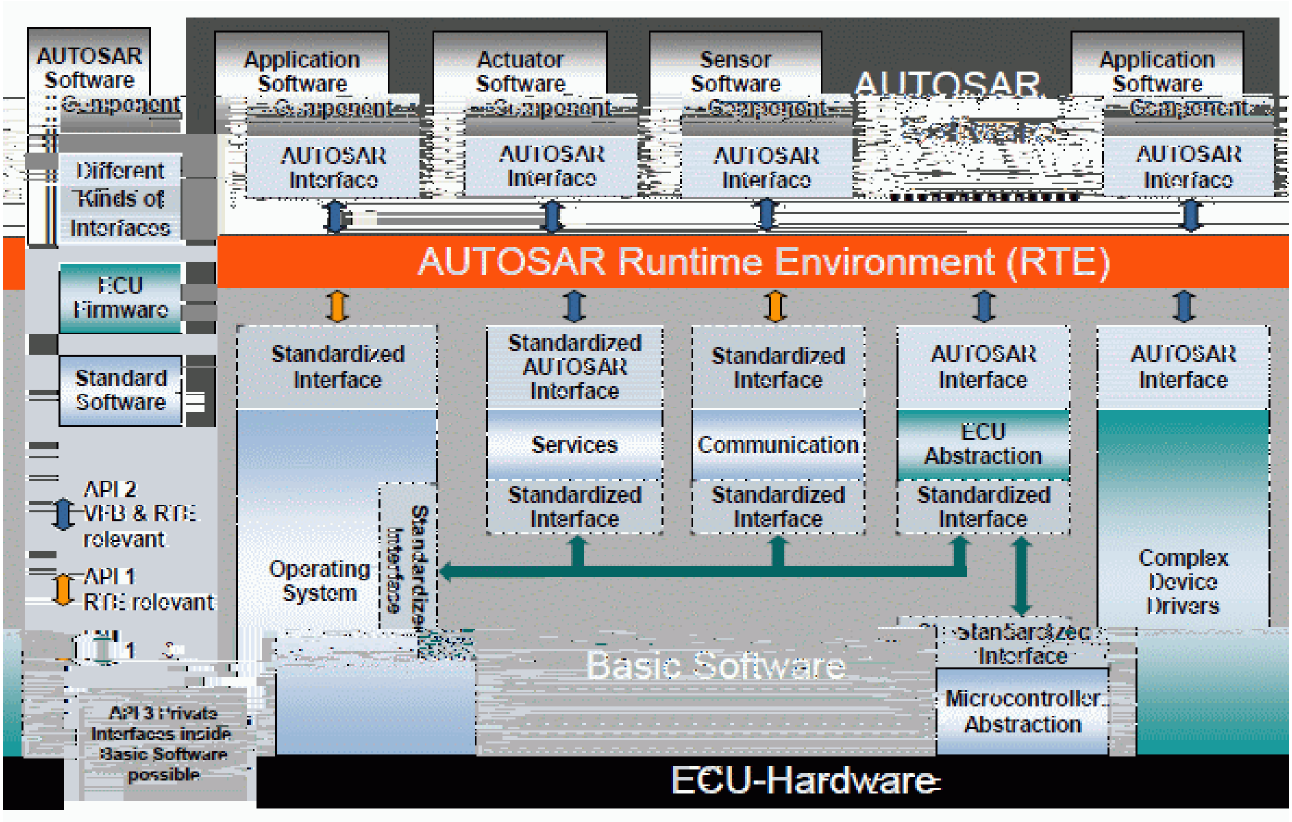


Figure 2-4: AUTOSAR software architecture[2]

2 Source of figure: AUTOSAR Technical Overview V2.2.2 R3.2 Rev 1

3 Measuring and Calibrating of ECU Software

In This Chapter You Will Find the Following Information:

3.1 Basics

Challenge

Variables in the source code are implemented as measurement and calibration parameters in the ECU software. The task of the calibration engineer is to measure and calibrate these parameters so that the behavior of the ECU is optimized. To make the calibration process convenient, calibration tools such as the MCD tool (**M**easurement, **C**alibration, **D**iagnostics) CANape are used. This type of tool requires an XCP driver and an A2L file for communicating with the ECU. The XCP driver enables the access to ECU-internal parameters during runtime. The A2L file, in turn, links the symbolic name of a measurement or calibration parameter with its memory address. In this way, the calibration engineer can calibrate individual calibration parameters with CANape without having to know the memory address of the parameter.

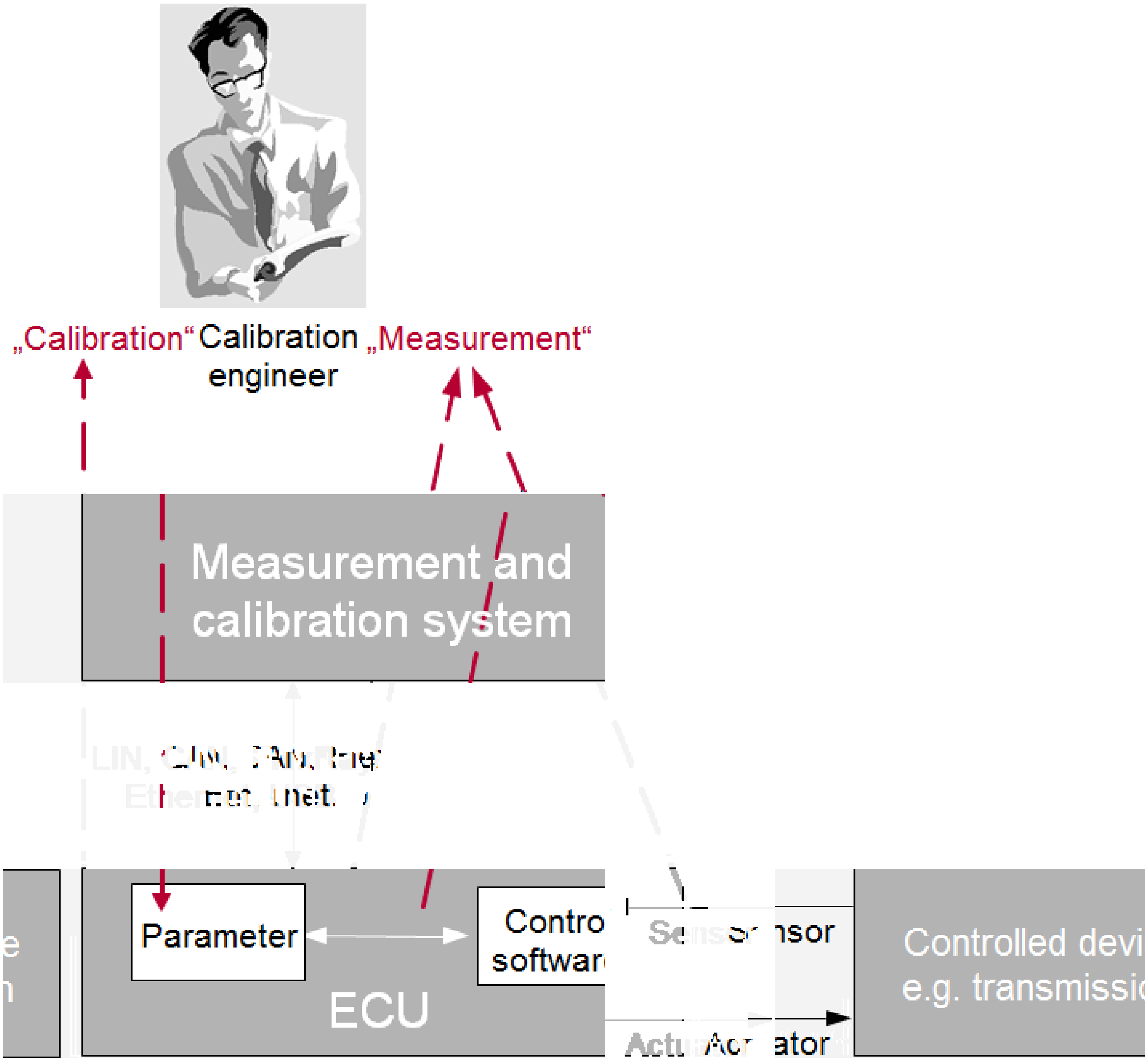


Figure 3-1: Measurement and calibration process

3.2 XCP Driver

Protocol

The XCP driver such as MICROSAR XCP is a further development of the CCP driver and can be used universally for different bus systems. It involves a protocol based on the single master/multi-slave principle. An XCP master, such as CANape, is able to communicate simultaneously with various XCP slaves. These include, for example, the ECU or HIL/SIL systems. Figure 3-2 shows the slave connection via XCP.

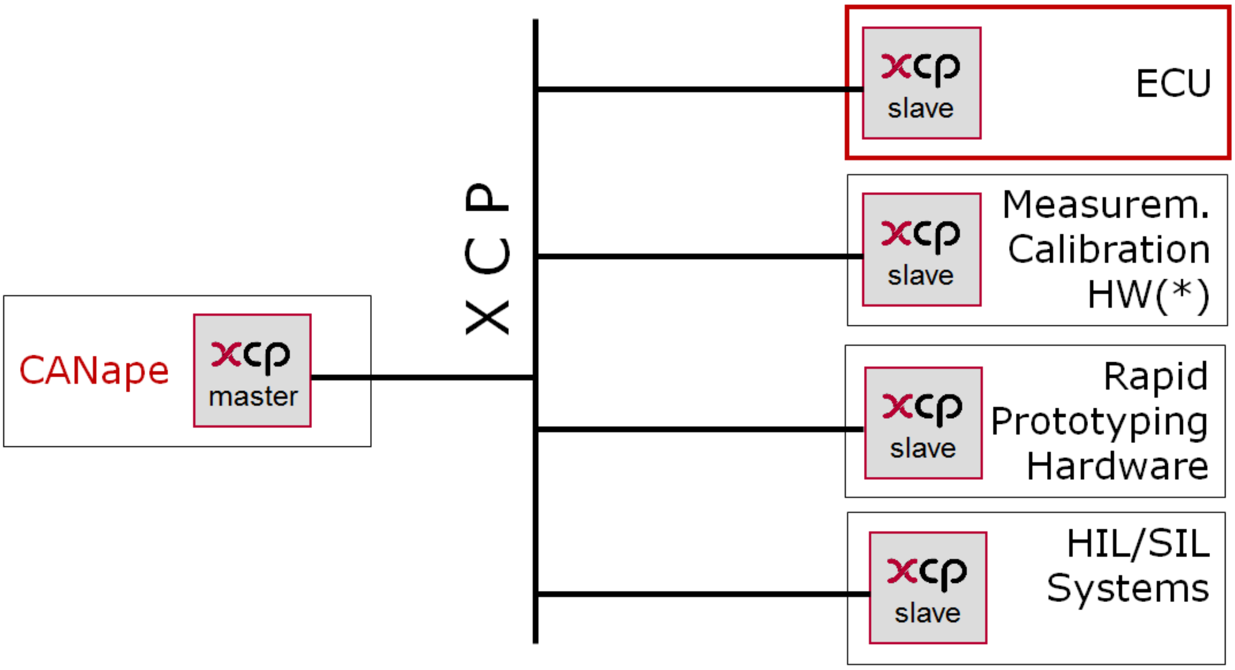


Figure 3-2: Communication possibilities of an XCP master such as CANape

Communication via A2L file

CANape communicates with the ECU via the XCP driver. The A2L file is an important component of this communication. From this file, the XCP master reads all information that is important for the communication setup and sequence.

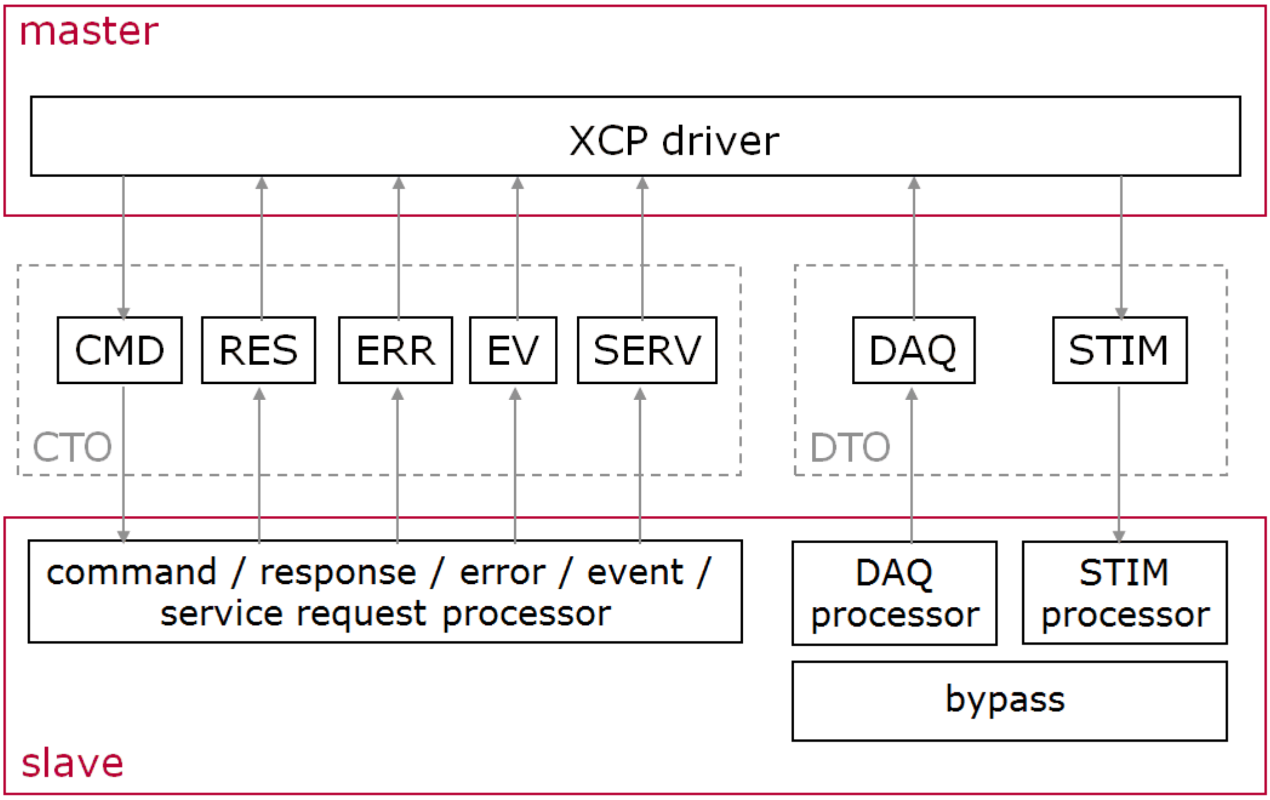


Figure 3-3: Single master/multi-slave concept

Transfer objects

In the XCP protocol, a distinction is made between "Command Transfer Objects (CTO)" and "Data Transfer Objects (DTO)" (see Figure 3-3). The Object Description Table (ODT) describes the mapping of the DTOs and memory of the slave. The reception of a CTO signals the slave to run a certain service. The transmission of a DTO is used for event-triggered reading and writing of objects from the memory of the XCP slave. For this, DAQ (Data Acquisition) lists are created from multiple ODTs in order to send the measurement values to the master at the same time that an event occurs. The events are defined using event channels and take over, with the help of defined time bases, the timing for task-synchronous transmission of measurement data.

Dynamic configuration of DAQ lists

With XCP it is possible to configure the DAQ lists both statically as well as dynamically. In the case of static configuration, the maximum number of DAQ lists, ODT tables, and ODT entries per DAQ list is fixed at compile time. With dynamic DAQ lists, on the other hand, only the maximum memory size is specified at compile time. This enables more efficient memory utilization since the size of the DAQ lists is defined individually. If necessary, it also allows more measurement signals to be measured compared to the static configuration. In addition, implementation in the XCP driver is significantly easier because specifications such as the maximum number of ODTs is eliminated. The dynamic configuration is therefore the only mode supported.

XCP features

The XCP protocol also enables use of some optional XCP features. These must be explicitly implemented and therefore be known to the supplier. The rest of this section presents the following XCP features in more detail: Measurement Modes, Autoselection and Software Version Check of the A2L File, Online Calibration, Page Switching, Bypassing and Resume Mode.

3.2.1 Measurement Modes

Measurement modes

The XCP protocol enables two different measurement modes: **Polling** and **DAQ** measurement. Both variants are briefly explained here.

Polling

Polling is the simplest measurement mode of the XCP protocol. In this mode, the XCP master uses an XCP command (`SHORT_UPLOAD`) to poll the measurement values in a uniform time base. The measurement data are not equidistant in this mode. If there is a high bus load, the measurement parameter may be transferred with a time lag. Figure 3-4 shows the communication sequence for the polling measurement mode.

[1] Trace

Time	Id	Name	Dir	Len	Data	Command
-0.400196s	Measurement started at					
0.01038s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.01327s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.10301s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.10506s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.20504s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.20744s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.30711s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.30965s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.4091s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.41145s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.50102s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.50285s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.60211s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.60362s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.70317s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.70557s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.80508s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.8078s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h
0.90707s	1	CMD	Tx	8	F4 01 00 00 03 10 20 00	SHORT_UPLOAD size=1 ext=0 addr=00201003h
0.9095s	2	RES	Rx	2	FF 00	Ok:SHORT_UPLOAD data=00h

*Figure 3-4: Communication sequence for the **polling** measurement mode*

DAQ

The **DAQ** measurement mode uses an optimized method in order to access ECU-internal values. In **DAQ** measurement mode, the XCP master groups the measurement and calibration parameters to be measured in ODTs and assigns these to the corresponding DAQ events before the start of the measurement. During the measurement, the XCP slave transmits the measurement values when the cyclic DAQ event or asynchronous DAQ event occurs without further requests to the master (see also the XCP Driver section).

In the **DAQ** measurement mode, a distinction is also made between the **Consistency ODT** and **Consistency DAQ** modes. In the first case, the measurement data of an ODT are consistent. In the second case, the DAQ list as a whole is consistent, but not every ODT as a single entity. The measurement data can therefore be split between two ODTs. Figure 3-5 presents the communication sequence of the **DAQ** measurement mode in a trace.

Note: Only the **Consistency ODT** is currently supported by MICROSAR.

[1] Trace

Time	Id	Name	Dir	Len	Data	Command
0.00032s	1	CMD	Tx	1	DC	GET_DAQ_CLOCK
0.00142s	2	RES	Rx	8	FF 00 6D 61 A6 A8 CC 0E	Ok:GET_DAQ_CLOCK timestamp=248293542
0.00154s	1	CMD	Tx	2	DD 01	START_STOP_SYNCH mode=01h
0.0024s	2	RES	Rx	1	FF	Ok:START_STOP_SYNCH
0.00801s	2	DAQ	Rx	7	00 00 A6 A8 CC 0E 00	Data
0.01803s	2	DAQ	Rx	7	00 00 0A A9 CC 0E 00	Data
0.02802s	2	DAQ	Rx	7	00 00 6E A9 CC 0E 00	Data
0.03803s	2	DAQ	Rx	7	00 00 D2 A9 CC 0E 00	Data
0.04804s	2	DAQ	Rx	7	00 00 36 AA CC 0E 00	Data
0.05824s	2	DAQ	Rx	7	00 00 9A AA CC 0E 00	Data
0.068s	2	DAQ	Rx	7	00 00 FE AA CC 0E 00	Data
0.07802s	2	DAQ	Rx	7	00 00 62 AB CC 0E 00	Data
0.08799s	2	DAQ	Rx	7	00 00 C6 AB CC 0E 00	Data
0.09799s	2	DAQ	Rx	7	00 00 2A AC CC 0E 00	Data
0.10805s	2	DAQ	Rx	7	00 00 8E AC CC 0E 00	Data
0.11807s	2	DAQ	Rx	7	00 00 F2 AC CC 0E 00	Data
0.12804s	2	DAQ	Rx	7	00 00 56 AD CC 0E 00	Data
0.13807s	2	DAQ	Rx	7	00 00 BA AD CC 0E 00	Data
0.14803s	2	DAQ	Rx	7	00 00 1E AE CC 0E 00	Data
0.15873s	2	DAQ	Rx	7	00 00 82 AE CC 0E 00	Data
0.168s	2	DAQ	Rx	7	00 00 E6 AE CC 0E 00	Data

Figure 3-5: Communication sequence for the ***DAQ*** *measurement mode*

Timestamps

If there are stringent requirements for time accuracy of the measurement values, generation of the timestamp directly in the ECU is recommended. In the DAQ measurement mode, the XCP driver also transfers the timestamp for each occurring event so that the measurement is not falsified by the running time of the transfer to the MCD tool. However, the throughput of measurement values is meanwhile reduced. Because the timestamps represent an additional load on the bus, their generation can also be controlled via the MCD tool.

With a CAN bus, for example, it should be possible to disable the timestamp. With Ethernet, the timestamp is of little importance.

Timestamps are mandatory on FlexRay when a cycle time is used that is faster than the FlexRay bus cycle.

3.2.2 Autoselection and Software Version Check of the A2L File

Software version check

CANape provides the option of checking the software version. This means that a check is made based on certain information to determine whether die A2L file integrated in CANape corresponds to the current software version of the connected ECU. The option also exists to select the A2L file automatically using an XCP protocol command.

CANape can use the following information for the software version check:

> XCP Station Identifier (protocol command `GET_ID`)

> EPK check

> Checksum of code segments in the ECU (CANape 11.0 and higher)

XCP station identifier

The "XCP Station Identifier" (`GET_ID`) represents the name of the A2L file during the software version check. This describes the software version in a meaningful way (e.g., `EcuName_V1-2-0.a2l`). CANape can use this identifier to check whether the correct A2L file is loaded or load the appropriate A2L file automatically.

EPK check

The EPK identifier (EPROM identifier) is a character string that is present in the ECU as well as in the database. The address in the ECU where this identifier can be found is specified in the database. This character string can, in turn, designate the software version based on the project name and its version.

Checksum The checksum of code segments (memory segments with ECU code) can be calculated for the HEX file and the ECU. On the basis of the checksum it can be determined if the HEX file, the A2L file, and the software on the ECU are compatible with respect to version. This approach assumes that the HEX file and the A2L file are viewed as a unit.

Application of the procedures The described procedures can be applied independently of one another. Each individual procedure increases the assurance that you are working with correct data. For example, it is possible to have the A2L file selected automatically based on the "XCP Station Identifier" and to additionally use the check based on EPK identifier.

3.2.3 Online Calibration

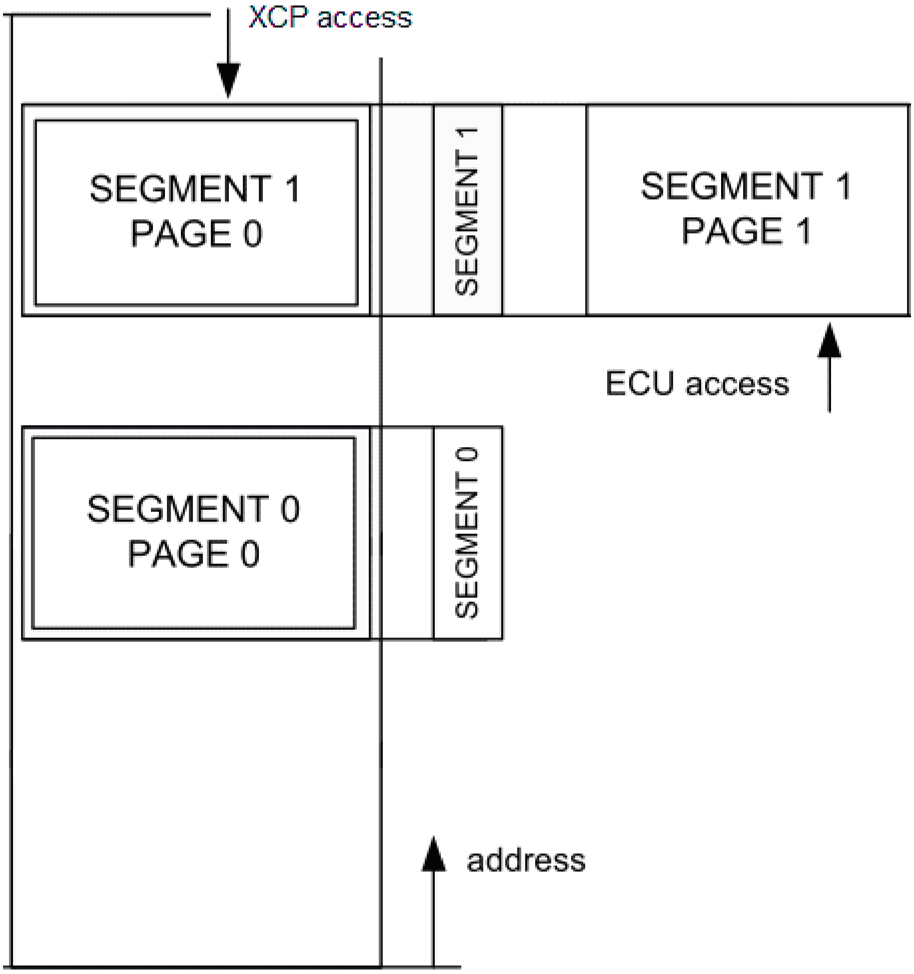
Prerequisite This section introduces the most important terms regarding online calibration. Online calibration enables optimization of the calibration parameters of the ECU algorithm during runtime so that the effects of the change can be directly measured. A prerequisite for this is availability of sufficient RAM memory.

Calibration concepts Two different calibration concepts are available for calibrating with XCP and AUTOSAR:

> InitRAM

> AUTOSAR Single Pointered

Figure 3-6: Physical layout of the memory



Assignment

The assignment of the algorithm to a page within a segment must be unambiguous at all times. In addition, only one page at a time may be active in a segment. This page is the so-called "active page for the ECU in this segment".

Access

The page that the ECU or the XCP driver accesses can be controlled individually. The df l h dj h i kh [F S df f h l fdohg kh df l h dj h i kh [F S df f h lq kl hj p hq 1

Commands

In order to use page switching, the ECU must support the XCP commands `GET_CAL_PAGE` and `SET_CAL_PAGE`.

With the `GET_CAL_PAGE` command, the master asks the slave which page of a segment is currently active. With the `SET_CAL_PAGE` command, on the other hand, the master can define which page the master itself or the ECU algorithm accesses.

3.2.5 Bypassing

Changes to the ECU algorithm

With the help of the bypassing feature, changes to the ECU algorithm can be made without calibration and subsequent flashing of the software.

Implementation

To implement bypassing, at least 2 XCP events as well as writable access to the ECU RAM via XCP are required. The events must differ in their direction (STIM, DAQ). Figure 3-7 shows the use of bypassing.

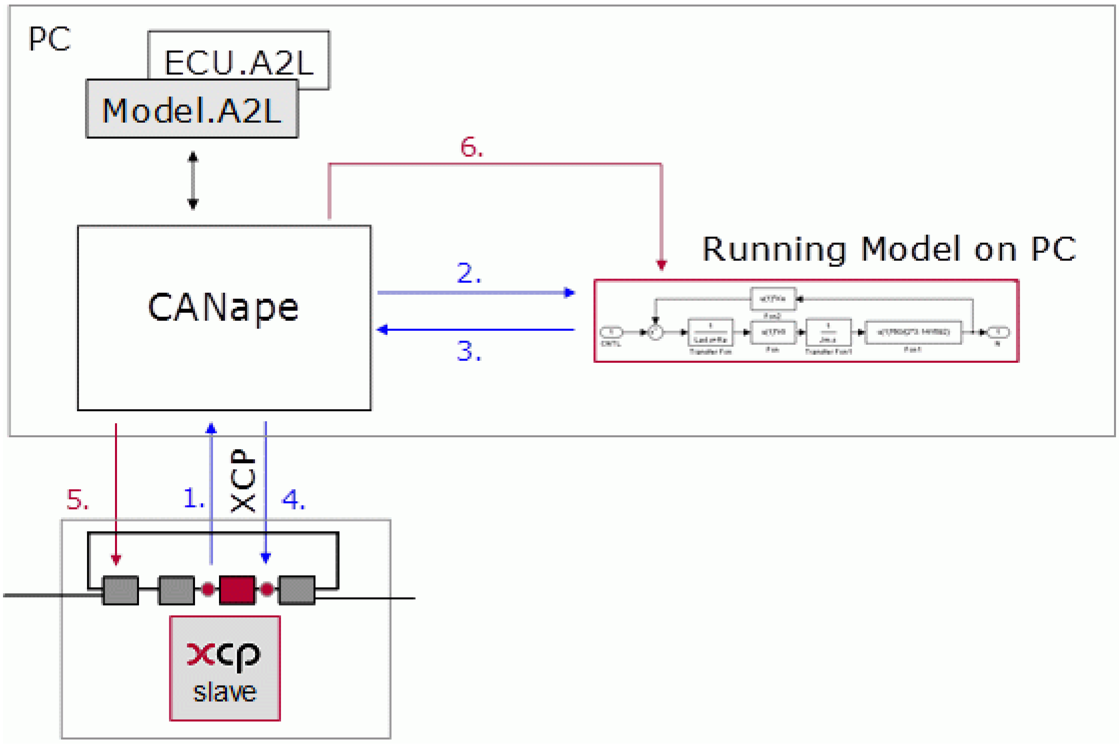


Figure 3-7: Use of bypassing

Signal path:

1. Reception of signals of the ECU (DAQ)
2. Transmission of signals as input of the model
3. Transmission of events back to the XCP master
4. Transmission of events back to the ECU (STIM)

Calibration path:

5. Calibration of the ECU (XCP)
6. Calibration of the model with XCP

Reference: The ASAM XCP Version 1.1 Part 1 - Overview specification, section 1.3 BYPASSING (BYP), explains in detail all other functions and implementations on the topic of bypassing.

3.2.6 Resume Mode

Automatic data transfer

Resume mode enables automatic data transfer to take place directly after switching on the ECU. This mode is commonly used to start recording and evaluating data as soon as the ECU starts. Resume mode supports both the STIM and DAQ directions. The `RESUME_SUPPORTED` flag in the DAQ properties must be set appropriately in the A2L file.

Commands

With the `START_STOP_DAQ_LIST` command (select), the master can select a DAQ list as part of a DAQ list configuration that the slave stores in non-volatile memory. The master then sends to the slave the configuration ID that the master has itself calculated and stored. The slave then knows that it will store the DAQ lists in non-volatile memory as soon as the `STORE_DAQ_REQ_RESUME` command is transmitted to it. The configuration ID is also stored in non-volatile memory so that the slave can return it upon the `GET_STATUS` command. Via the `GET_STATUS` command, the master finds out whether a slave is in resume mode. Prior to storing, the slave deletes the previous content of the non-volatile memory.

After each start of the slave, it sends the `EV_RESUME_MODE` command to the master. This command contains the following data:

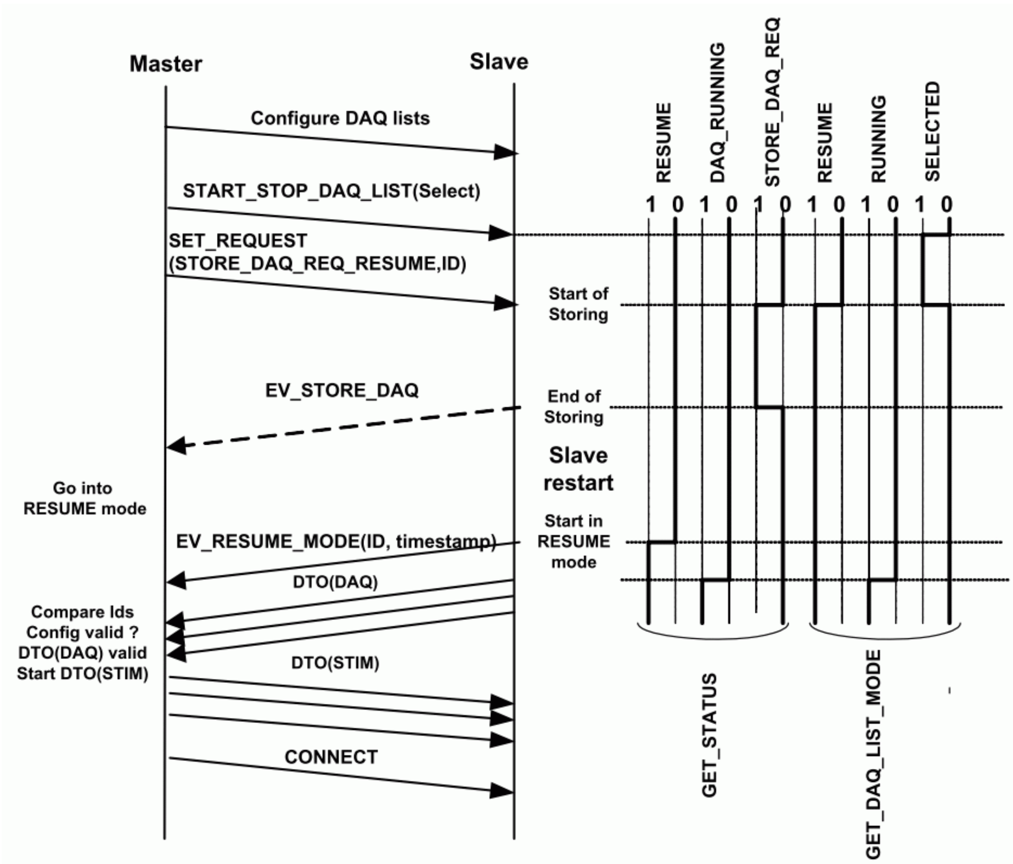
Figure 3-8: Data of the `EV_RESUME_MODE` command

Position	Type	Description
0	BYTE	Packet ID: Event 0xFD
1	BYTE	[illegible] EV_RESUME_MODE [illegible]
2,3	WORD	Session Configuration Id from slave
4..7	DWORD	Current slave Timestamp (optional)

Communication sequence

The communication sequence between the master and slave can be tracked in Figure 3-9.

Figure 3-9: Communication sequence between master and slave



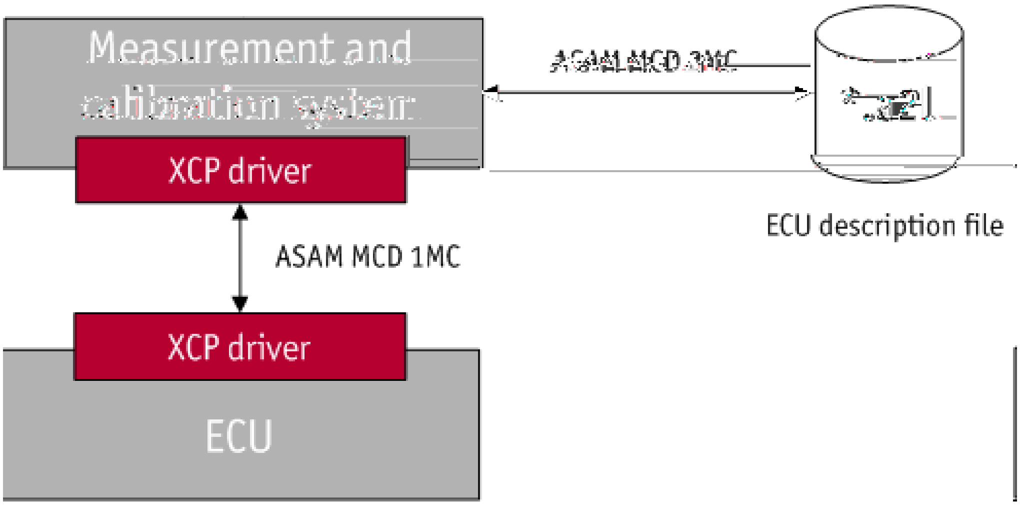
Reference: Additional XCP commands and information regarding resume mode can be found in the ASAM XCP Version 1.1 Part 1 - Overview specification.

3.3 A2L File

Goal

The A2L file has been specified by the Association for Standardization of Automation and Measuring Systems (ASAM) with the goal of defining compatible and replaceable modules for electronics development in the automotive industry.

Figure 3-10: ASAM interfaces



ECU description file

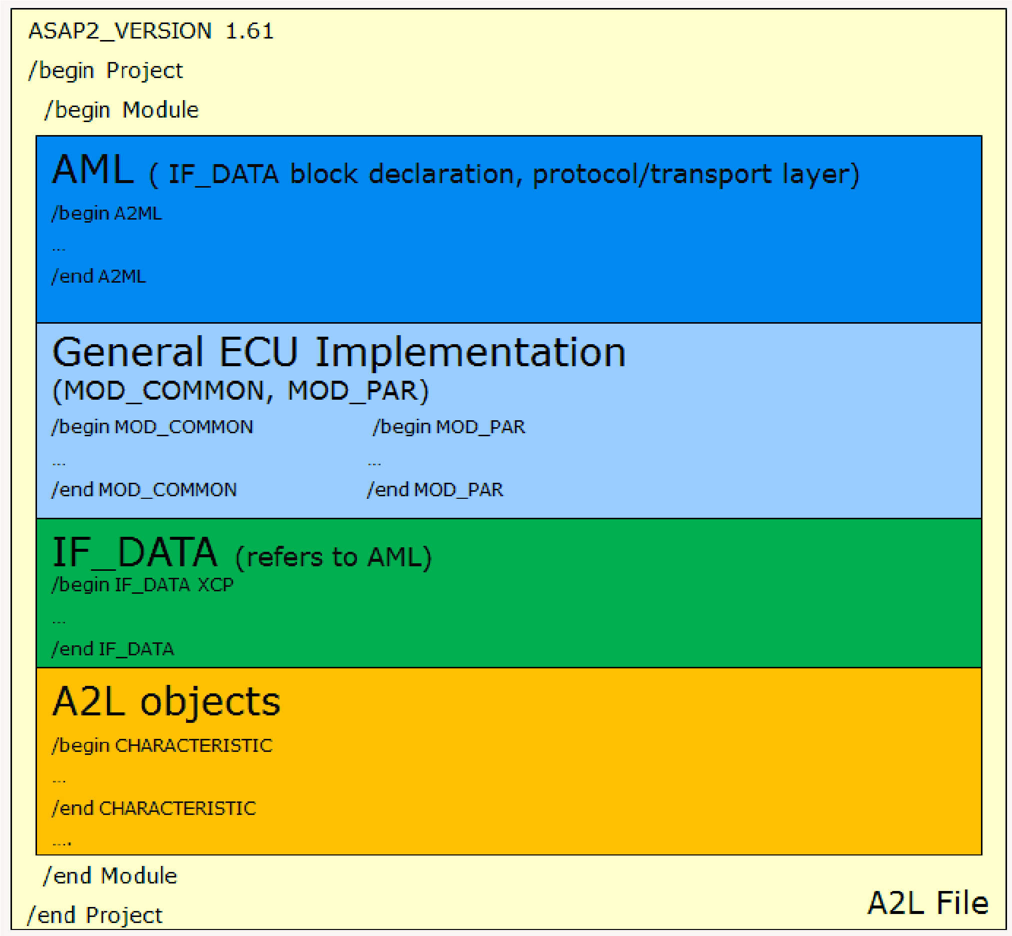
The description file of the ECU for configuring the models and the layout of the calibratable and measurable objects supplies the ASAP2 (ASAM MCD 2MC) interface in the form of the A2L file. Finally, the data exchange between the MCD system and the ECU is specified via the ASAM MCD 1MC (ASAP1b) interface.

3.3.1 Structure

Modular structure

The A2L file has a modular structure, which enables the replacement of individual modules without having to adapt the entire A2L file. Figure 3-11 shows this modular structure.

Figure 3-11: Structure of the A2L file



4 major parts

The project-relevant data at the start of the A2L file are defined using the `PROJECT` keyword and form the framework of the A2L file. These also include the ECU description that can be described with the `MODULE` keyword and divided into 4 major parts:

> AML
> General ECU Implementation
> IF_DATA
> A2L Objects

These parts are explained in more detail below.

3.3.1.1 AML

Interface-specific parameters

The first part defines the interface-specific parameters. It yields the framework of the `IF_DATA` area that is defined using the A2ML metalanguage with the `AML` keyword. The AML is generally configured once since the specification of a driver and the corresponding features is also performed once.

Reference: Detailed information regarding the metalanguage can be found in the ASAP2 specification ASAM MCD-2 MC, chapter 5.

3.3.1.2 General ECU Implementation

ECU description

This part of the A2L file specifies the ECU description. Here, standardized structures of the ECU and the general description are defined using the `MOD_COMMON` and `MOD_PAR` keywords. This part of the A2L file also generally remains unchanged since the structures of the ECU are set. The keywords are now briefly presented:

The `MOD_COMMON` keyword describes the internal structures of the ECU. The possibility exists to define certain parameters for the complete ECU. For example, if a standard byte order exists, this can be specified for the complete device in this area.

Example:

```
/begin MOD_COMMON ""
  BYTE_ORDER MSB_LAST
  …
/end MOD_COMMON
```

The `MOD_PAR` keyword describes the ECU-specific description data such as the EPROM identifier or the memory segments.

Example:

```
/begin MOD_PAR "Comment"
  ADDR_EPK 0x12345
  EPK "EPROM identifier test"
  /begin MEMORY_SEGMENT  Data0001 "Data segment" DATA
   FLASH INTERN 0x30000 0x1000 -1 -1 -1 -1 -1
  /end MEMORY_SEGMENT
  SYSTEM_CONSTANT "CONTROLLERx CONSTANT1" "0.99"
/end MOD_PAR
```

3.3.1.3 IF_DATA

Communication interface

Next, the communication interface is specified using the `IF_DATA` keyword. This is only adapted if, for example, certain XCP commands are also to be used afterwards.

The `IF_DATA` keyword describes the interface-specific data, such as protocol layer or DAQ lists. These can also be defined directly as a subcategory for diverse A2L objects.

Example:

```
/begin IF_DATA XCP
  /begin PROTOCOL_LAYER
    …
  /end PROTOCOL_LAYER
  /begin DAQ
    …
  /end DAQ
  /begin XCP_ON_CAN
    …
  /end XCP_ON_CAN
/end IF_DATA
```

DAQ configuration

The DAQ configuration is an essential component of the XCP protocol and will therefore be presented again in more detail. The configuration is made under the `DAQ` keyword in the `IF_DATA` section, and the individual events are defined under this point.

Reference: More detailed information regarding the definition of events can be found in the ASAM XCP Version 1.1 Part 1 - Overview specification, section 1.1.1.5 Event Channels.

Specification of DAQ lists

Table 3-1: Specification of DAQ lists in the `IF_DATA` section compares the static and dynamic DAQ configuration in the A2L file.

XCP (static)	XCP (dynamic)	Explanation
/begin DAQ	/begin DAQ	
STATIC	DYNAMIC	DAQ configuration
RESUME_SUPPORTED	RESUME_SUPPORTED	Resume mode is supported
/begin DAQ_LIST		
0x0		DAQ list number
DAQ_LIST_TYPE DAQ		Direction (DAQ | STIM)
MAX_ODT 0xB		Maximum ODTs
MAX_ODT_ENTRIES 0x7		Maximum entries in an ODT
FIRST_PID 0x3		Packet designator
EVENT_FIXED 0x0		Event channel is permanently specified
/end DAQ_LIST		
/begin EVENT	/begin EVENT	
"10 ms Liste 1"	"10 ms Liste 1"	Name of the event channel
"10 ms Lis"	"10 ms Lis"	Brief name of the event channel
0x0000	0x0000	Number of the event channel
DAQ	DAQ	Direction (DAQ | STIM)
0x01	0x01	Maximum of DAQ lists
0x0A	0x0A	Sampling period (0 corresponds to non-cyclic)
0x06	0x06	Time base(0x06 corresponds to 1ms)
0x00	0x00	Priority
/end EVENT	/end EVENT	
/end DAQ	/end DAQ	

Table 3-1: Specification of DAQ lists in the `IF_DATA` section

Default event

It is recommended to assign at least one default event to each measurement and calibration parameter in order to ensure that the objects will be measured at the correct time in each case (example in next section under A2L Objects | Measurement parameters). With the help of this assignment, the drag & drop feature of the display windows in CANape can be used optimally. If a default event is not defined, the measurement mode must be changed manually by polling the appropriate event.

3.3.1.4 A2L Objects

Specification of parameters and keywords

The last part contains the A2L objects. The measurement and calibration parameters are specified here using various parameters and keywords. In this area, changes may occur even after completion of the A2L file since, for example, measurement parameters will also be added during the course of the project.

Measurement parameters

The measurement parameters are defined using the `MEASUREMENT` keyword. Some parameter values are optional (labeled in `[]`), while other values, such as the name, are mandatory.

Prototype:

```
/begin MEASUREMENT
    ident  Name
    string  LongIdentifier
    datatype  Datatype
    ident  Conversion
    uint  Resolution
    float  Accuracy
    float  LowerLimit
    float  UpperLimit
    [-> ANNOTATION]*
    [-> ARRAY_SIZE]
    [-> BIT_MASK]
    [-> BIT_OPERATION]
    [-> BYTE_ORDER]
    [-> DISCRETE]
    [-> DISPLAY_IDENTIFIER]
    [-> ECU_ADDRESS]
    [-> ECU_ADDRESS_EXTENSION]
    [-> ERROR_MASK]
    [-> FORMAT]
    [-> FUNCTION_LIST]
    [-> IF_DATA]*
    [-> LAYOUT]
    [-> MATRIX_DIM]
    [-> MAX_REFRESH]
    [-> PHYS_UNIT]
    [-> READ_WRITE]
    [-> REF_MEMORY_SEGMENT]
    [-> SYMBOL_LINK]
    [-> VIRTUAL]
/end MEASUREMENT
```

Example:

```
/begin MEASUREMENT
  FP_LED
  "Raw value target driving program"
  UBYTE NonDim_2p0 0 0 0 10
  ECU_ADDRESS 0xD000B47C
  ECU_ADDRESS_EXTENSION 0x0
  /begin IF_DATA XCP
    /begin DAQ_EVENT VARIABLE
      /begin DEFAULT_EVENT_LIST
        EVENT 0001
      /end DEFAULT_EVENT_LIST
    /end DAQ_EVENT
  /end IF_DATA
/end MEASUREMENT
```

Calibration parameters

The calibration parameters are specified in the A2L file using the `CHARACTERISTIC` keyword. In this case, as well, there are optional parameter values `[]` and mandatory parameter values.

```
Prototype:
      /begin CHARACTERISTIC  ident  Name
          string  LongIdentifier
          enum  Type
          ulong  Address
          ident  Deposit
          float  MaxDiff
          ident  Conversion
          float  LowerLimit
          float  UpperLimit
          [-> ANNOTATION]*
          [-> AXIS_DESCR]*
          [-> BIT_MASK]
          [-> BYTE_ORDER]
          [-> CALIBRATION_ACCESS]
          [-> COMPARISON_QUANTITY]
          [-> DEPENDENT_CHARACTERISTIC]
          [-> DISCRETE]
          [-> DISPLAY_IDENTIFIER]
          [-> ECU_ADDRESS_EXTENSION]
          [-> EXTENDED_LIMITS]
          [-> FORMAT]
          [-> FUNCTION_LIST]
          [-> GUARD_RAILS]
          [-> IF_DATA]*
          [-> MAP_LIST]
          [-> MATRIX_DIM]
          [-> MAX_REFRESH]
          [-> NUMBER]
```

```
        [-> PHYS_UNIT]
        [-> READ_ONLY]
        [-> REF_MEMORY_SEGMENT]
        [-> STEP_SIZE]
        [-> SYMBOL_LINK]
        [-> VIRTUAL_CHARACTERISTIC]
    /end CHARACTERISTIC
```

Example:

```
/begin CHARACTERISTIC Pehp_IDATA.T_FP_delay
  "Time for transition from target to actual driving program
  HPP"
  VALUE 0xA01350CC UWORD_COL_DIRECT 0 ms_f10 0 60000
  ECU_ADDRESS_EXTENSION 0x0
  EXTENDED_LIMITS 0 60000
  BYTE_ORDER MSB_LAST
  FORMAT "%6.0"
/end CHARACTERISTIC
```

Conversion rules

Frequently, conversion rules are additionally defined for measurement or calibration parameters if, for example, an object is to be converted to a physical unit. The `COMPU_METHOD` keyword is used for this.

Prototype:

```
/begin COMPU_METHOD  ident  Name
    string  LongIdentifier
    enum  ConversionType
    string  Format
    string  Unit
    [-> COEFFS]
    [-> COEFFS_LINEAR]
    [-> COMPU_TAB_REF]
    [-> FORMULA]
    [-> REF_UNIT]
    [-> STATUS_STRING_REF]
/end COMPU_METHOD
```

There are various conversion types for this:

IDENTICAL Raw value and physical value are identical, no conversion is necessary

FORM A formula is used for the conversion (to be specified with the `FORMULA` keyword)

LINEAR Conversion is made linearly according to `f(x)=ax+b`
(a and b are specified using the `COEFFS_LINEAR` keyword)

RAT_FUNC Conversion is made using a rational function:
`f(x) = (axx+bx+c)/(dxx+ex+f)`
`a, b, c, d, e, f` are specified using the `COEFFS` keyword.

TAB_INTP Conversion table with interpolation

TAB_NOINTP Conversion table without interpolation

TAB_VERB Verbal conversion table

Example:

```
/begin COMPU_METHOD NonDim_2p0_a ""
  RAT_FUNC "%5.0" "-"
  COEFFS 2 1 0 0 4 1
/end COMPU_METHOD
```

Groups

Hierarchy levels are realized in the A2L file using groups. In a project with many measurement and calibration parameters, these can be subdivided and categorized. The possibility also exists to define subgroups. This makes the A2L file easier to view in CANape.

Prototype:

```
   /begin GROUP  ident  GroupName
       string  GroupLongIdentifier
       [-> ANNOTATION]*
       [-> FUNCTION_LIST]
       [-> IF_DATA]*
       [-> REF_CHARACTERISTIC]
       [-> REF_MEASUREMENT]
       [-> ROOT]
       [-> SUB_GROUP]
   /end GROUP
```

Example:

```
/begin GROUP Maps "Calibration Maps"
 ROOT
  /begin SUB_GROUP
    WorkingPoint
  /end SUB_GROUP
  /begin REF_CHARACTERISTIC
    KF1 KF2 KF3 KF4 KF5 KF6 KF7 KF8
    TestKennfeld map1_8_8_uc map4_80_uc map5_82_uc
  /end REF_CHARACTERISTIC
/end GROUP
```

Structures

The A2L Specification contains no keyword for structures. CANape identifies these based on analysis of the object name.

The valid syntax for structures in the A2L has the following appearance:

"`.`" for objects (e.g., "`TestStructStruct1.TestStruct2.s1`")

"`[]`" for arrays (e.g., "`TestStructStruct1.TestStruct2.s1[0]`")

Example:

```
/begin CHARACTERISTIC Test1.s0 ""
  VALUE 0x2080D0 __ULONG_S 0 Test1.s0.CONVERSION 0 4294967295
  ECU_ADDRESS_EXTENSION 0x0
  EXTENDED_LIMITS 0 4294967295
  FORMAT "%.15"
/end CHARACTERISTIC
```

Reference: Detailed information on the meaning of individual parameters can be found in the ASAP2 specification ASAM MCD-2 MC under the respective keyword.

3.3.2 Mode of Functioning

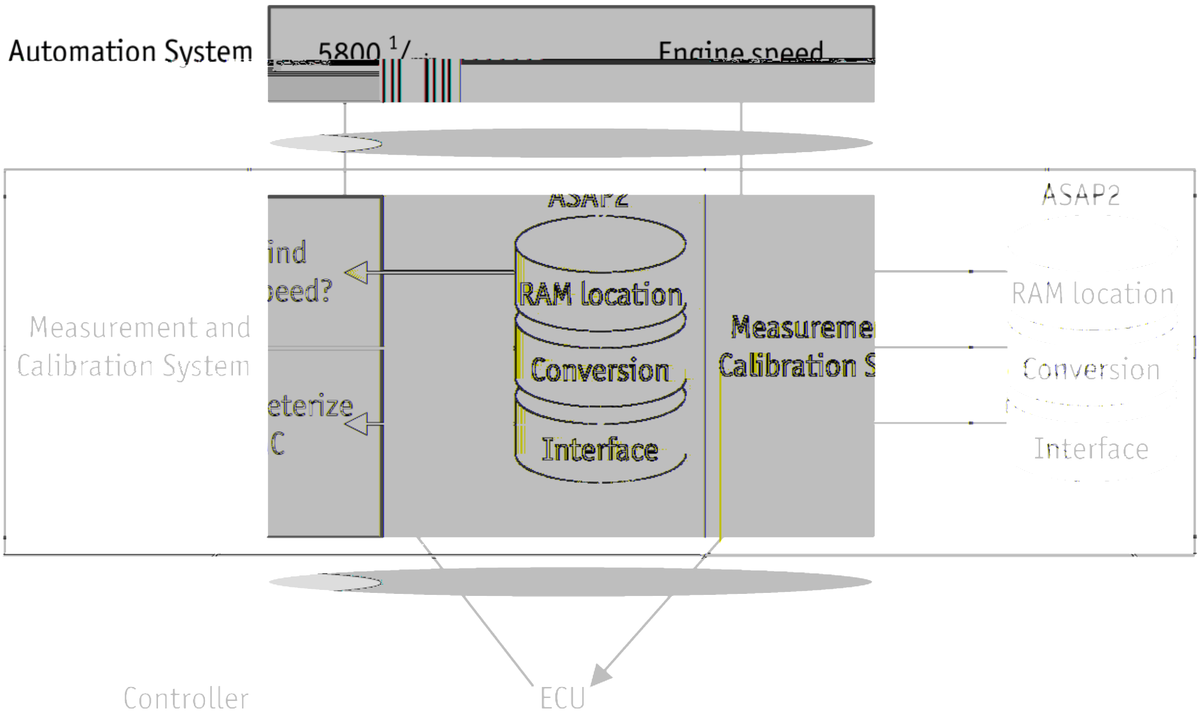


Figure 3-12: Mode of functioning of the A2L

Engine speed as example

Figure 3-12 illustrates the mode of functioning of an A2L file. The engine speed is read out here as an example. Via the A2L file, the measurement and calibration system (CANape) learns which memory address contains the engine speed and how the ASAM MCD 1MC interface must be parameterized. The read-out raw value is then converted to a physical value using a conversion rule described in the A2L file.

4 OEM

In This Chapter You Will Find the Following Information:

4.1 Objective

Checklist for performance specifications

This chapter serves as a checklist for creating performance specifications. The OEM must ensure that the indicated items are incorporated in the performance specifications after careful consideration and as needed.

4.2 Content of the Performance Specifications

The content of the performance specifications must define the desired requirements for the supplier. These can be divided into mandatory and optional requirements.

Mandatory Requirements

> Delivery of an A2L compatible with the software version
> Configured XCP driver
> Preconfigured CANape project

Optional Requirements

> Build environment that can generate the A2L
> Delivery of a linker MAP file

The delivery of a linker MAP file has the advantage that new measurement and calibration parameters can be incorporated directly into the A2L file. If a request to measure additional objects arises during the course of a measurement task, the memory addresses are known and these can be added.

4.3 Measurement Task

Information to be communicated

To realize the mandatory requirements relating to the measurement task, the supplier requires some information, such as the category of the measurement parameters. Various details about the DAQ configuration are also relevant both for the configured XCP driver and for the A2L file.

Specifically, information on the following items must be communicated to the supplier:

> Category of the measurement parameters
>> Software component
>> Basic software
>> BSW module (e.g., COM, CanNm)
>> Runtime monitoring
> Event-triggered measuring via DAQ
>> Static or dynamic DAQ lists
>> Vector recommends dynamic DAQ lists in order to make more efficient use of the memory and, if necessary, to allow more signals to be measured.
>> Definition of DAQ event time base
>> Use of timestamps
>> Use of DAQ default events

4.4 Calibration Task

Items to be considered

The calibration task also affects the mandatory requirements (A2L file, XCP driver) for the supplier. The following items should be carefully considered here:

- Location of the calibration parameters
  - Software component
  - NVRAM
- Optimized "going online"

  The accesses to the ECU are decreased when "going online" is optimized. An upload operation is performed only if differences between the data in the memory image and the ECU are identified. This procedure accelerates the "going online".

  For optimized "going online", the use of a memory image is required. The memory image is described on the basis of memory segments, which contain only calibration parameters. In addition, the checksum calculation must be implemented in the ECU.

  Optimized "going online" is also a prerequisite for offline calibration and the use of dataset management.
- Use of a flashable HEX file (with calibrated calibration parameters from CANape)

4.5 XCP Features

Features to be supported

The XCP Driver section explained some aspects and features of the XCP protocol. Specifically, the following were explained: Measurement Modes, Autoselection and Software Version Check of the A2L File, Online Calibration, Page Switching, Bypassing, and Resume Mode. It is important here that the OEM communicates to the supplier which of these features are to be supported by the XCP driver.

It is recommended to incorporate the following XCP features in the performance specifications:

- Polling and DAQ measurement modes
- Autoselection of A2L and the software version check
- Online calibration
- Resume mode

5 Supplier

In This Chapter You Will Find the Following Information:

5.1 Preface

Certain functions and configurations for AUTOSAR

The measurement and calibration task assigned by the OEM is carried out in the implementation of the ECU software. When AUTOSAR-compliant software modules are used, the modules must be configured appropriately and certain functions must be implemented.

This chapter explains procedures for implementing the requirements for the ECU software. The description refers to the MICROSAR product.

The first part describes the configuration of software components (SW-C), the MICROSAR RTE, and the MICROSAR BSW module.

This is followed by a brief overview of the integration of the XCP slave. The XCP slave is provided by the XCP module.

The final part describes the creation of the A2L description file, which will be a central component of the CANape configuration.

5.2 Requirements

Software components

The following software components at least starting with the following versions are required for the descriptions:

> Vector Informatik DaVinci Developer 3.0.110 (SP5)
> Vector Informatik DaVinci Configurator 4.1.1.2
> Vector Informatik ASAP2 Tool-Set 7.0
> Vector Informatik CANape 10.0 SP4
> Vector MICROSAR Basic Software starting with Release 14 including MICROSAR XCP and MICROSAR RTE

5.3 Definition of Measurement and Calibration Parameters

Via software components

The measurement and calibration parameters for the measurement and calibration task of the OEM are usually located in the software components (SW-C). These parameters are defined with configuration tools, such as the DaVinci Developer. Configuration of the RTE is also required for this.

Without RTE

Other measurement and calibration parameters can also be provided without the support of AUTOSAR interfaces. A brief explanation is given in the Measuring and Calibrating Without the Support of the RTE section.

Via A2L file

In addition, measurement parameters can also be added to the measurement configuration within the AUTOSAR basic software. This is done by inserting known measurement parameters from the basic software into the A2L file.

5.3.1 Measuring and Calibrating of AUTOSAR Software Components

Measurable objects

Measurable objects can be configured using a configuration tool, such as the DaVinci Developer. Measurable objects include data elements ( ) of application and service ports ( ), variables for communication between runnables ( ), and calibration parameters ( ).

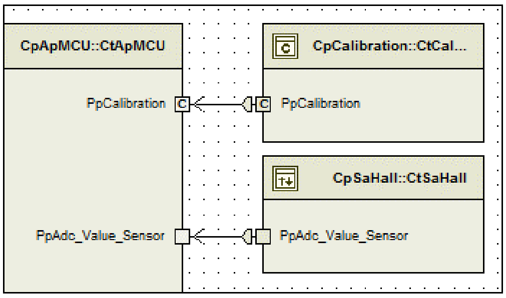


Figure 5-1: SW-C connected to ports

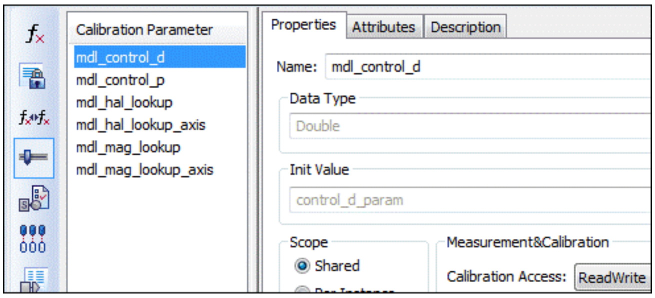
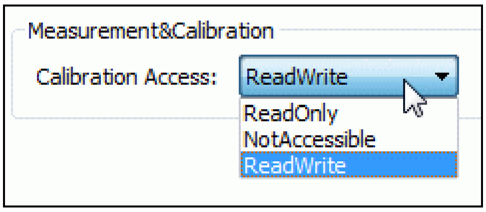


Figure 5-2: SW-C with parameters for measuring/calibrating

Calibration access

The objects indicated above can be made measurable by setting **Calibration Access** to **ReadOnly** in the DaVinci Developer. The **ReadWrite** setting enables the writing of objects with CANape. The writing of calibration parameters occurs in the common "Calibration" use case of CANape. The writing of other data elements can be configured but is not recommended. This is because the write access is not exclusive, which means that information can be overwritten again.

Figure 5-3: Measurement and calibration option for an object (e.g., `data element`*)*



Specifying calibration parameters

The AUTOSAR Standard provides the option of specifying calibration parameters. Two variants are differentiated.

Calibration parameters can be defined within a software component. These are then also available only for this software component.

The second variant is the use of a calibration software component that can provide calibration parameters for multiple software components.

5.3.2 Measuring of Ports and Variables

Configuration of the data elements

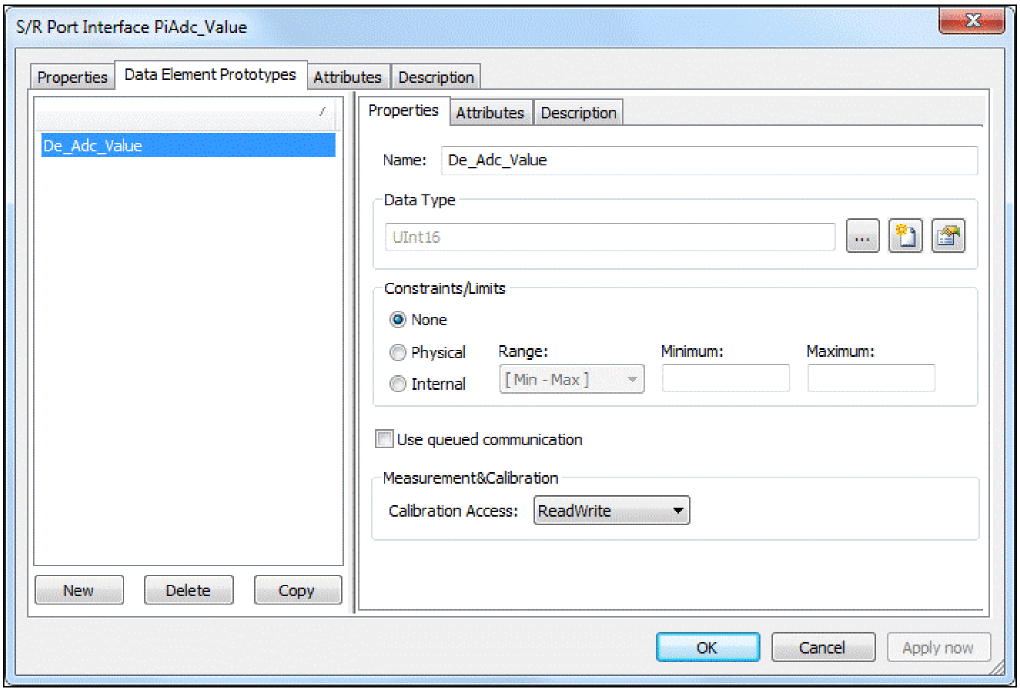
Data elements to be measured must be configured appropriately with the help of the **Calibration & Measurement Support**. For measuring, **Calibration Access** must be set to **ReadOnly**.

The following data elements can be measured:

-
-
-
-

For and , the data elements can be easily configured for calibrating via the **Properties**.

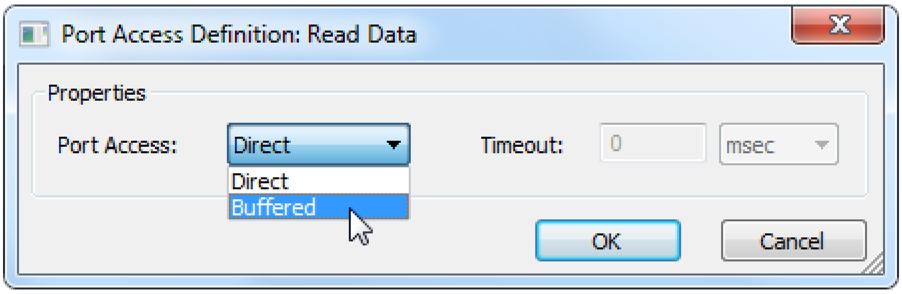
Figure 5-4: Configuration of a sender/receiver port



Special case: Data Mapping

for which a Data Mapping is defined represent a special case. For these ports, a direct (explicit) access and a buffered (implicit) access can be configured as shown in Figure 5-5.

Figure 5-5: Access definition of a port



Ports that have explicit access configured can only be measured using the BSW module COM. On the other hand, ports whose access was configured as buffered can be measured using the RTE as well as the BSW module COM.

The measurement parameters are typically already preconfigured.

5.3.3 XCP Events

RTE support

The RTE supports the generation of XCP events. For one thing, an event is created for each task. These events are used to measure variables of the runnables that are run within the task. The following should be noted in this regard:

> The RTE generates XCP events at the end of each task. An XCP event thus does not have a direct relation to the running of a Runnable. It is therefore common that a Runnable does not run continuously between XCP events.

> If XCP events are generated by the RTE, the DAQ measurement mode must also be activated in the XCP module.

> It must thereby be anticipated that the XCP events of the RTE will be called very often.

> The generated XCP events are not cyclic, so it is not possible to make a definitive statement about the expected bus load.

For another thing, the RTE also generates XCP events for the above-mentioned access to buffered ports. By means of the description in the A2L file, it is ensured that these ports are measured fixed with the generated event.

5.3.4 Software Component with Calibration Parameters

External access

The definition of calibration parameters ( ) makes it possible to change a calibration parameter within the software component externally via XCP.

Within the software component, access to this calibration parameter is read-only. However, outside of the SW-C, the possibility exists to change this calibration parameter.

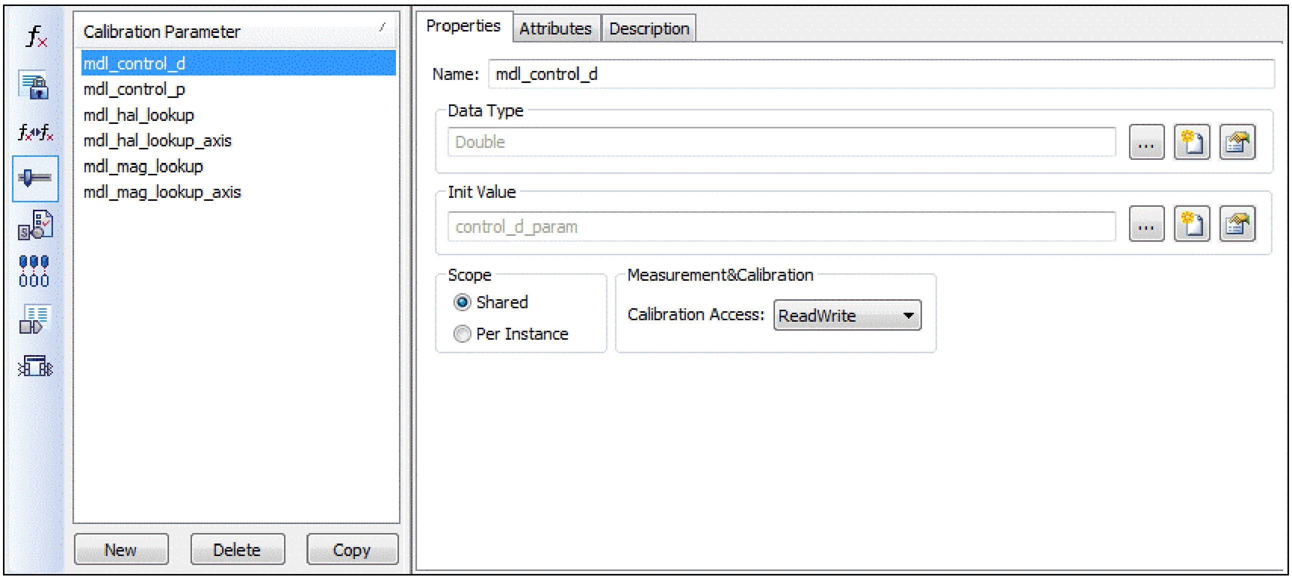


Figure 5-6: Properties of a calibration parameter

A calibration parameter consists of a data type and an initial value. The scope (**Scope**) and the measurement and calibration access can be configured.

Note: For additional information about these parameters, refer to the online help for the DaVinci Developer.

5.3.5 Calibration Parameters for Multiple Software Components

Calibration-type software component

A calibration-type software component is used to provide calibration parameters for multiple software components.

This type of software component has only calibration ports ( ), that provide calibration parameters for other SW-Cs and act as a sender port.

Representation of a calibration software component in DaVinci Developer:

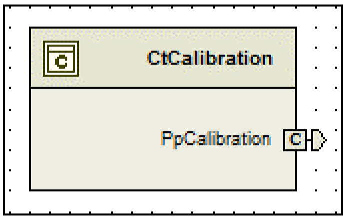


Figure 5-7: Graphic interface

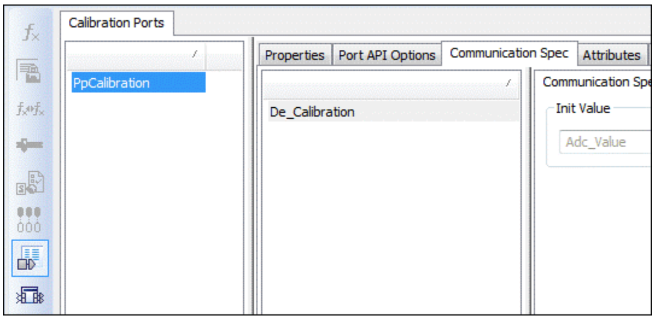


Figure 5-8: List with the configured ports

Each calibration port, in turn, contains calibration parameters. These calibration parameters are handled in just the same way as calibration parameters within a software component.

5.3.6 Configuration of the RTE (Runtime Environment)

RTE support necessary

The support of the RTE is required in order to measure and calibrate software components using the XCP protocol. The MICROSAR RTE Generator provides this measurement and calibration support.

Reference: For an explanation of the activation of the measurement and calibration support, refer to the technical reference TechnicalReference_Asr_Rte, page 102ff.

Online calibration procedures supported by CANape

CANape currently supports the following online calibration procedures:

> Initialized RAM
> Single Pointered

Initialized RAM

The standard calibration procedure with CANape is "Initialized RAM". This procedure is suitable when the ECU has sufficient RAM memory for all calibration parameters to be calibrated.

Single Pointered

The advantage of the "Single Pointered" calibration concept is that not all calibration parameters constantly have a copy in the RAM memory. Therefore, this procedure must be chosen when RAM memory capacity is limited.

When the ECU source code is generated by the DaVinci Developer, A2L fragments are also generated. The integration of the created A2L fragments `Rte.a2l` and `Rte_XcpEvents.a2l` is described in more detail in the Creating an A2L File section.

5.3.7 Measuring and Calibrating Without the Support of the RTE

Points to be considered

The possibility exists to use measurement and calibration even without the support of the RTE.

The following points must be noted in this regard:

> Measurement via DAQ events requires that corresponding XCP events be programmed and then described in the A2L file.

Example: Integrating an XCP event within a runnable

```
FUNC(void, RTE_CTAPMCU_APPL_CODE) RCtApMy_Algo(void)
{
  // Perform algorithm within my runnable
  ...

  // Trigger user defined XCP Event
  XcpEvent(12);
}
```

> For online calibration, a separate implementation of the calibration method ("Initialized RAM" or AUTOSAR "Single Pointered") is required.

> Calibration and measurement requires one or more A2L files that are created manually or with an external program (e.g., ASAP2 Creator or TargetLink). These A2L files must be merged with the A2L files generated by the Vector tools. The ASAP2 Merger program can be used for this (see description in the Creating an A2L File section on page 49).

5.3.8 Debugging of the BSW (Basic Software)

Modules which provide measurement parameters

MICROSAR AMD allows measuring BSW internal status information using XCP in order to ease debugging. For this purpose MICROSAR AMD provides measurement parameters for by different MICROSAR modules such as COM, CANNM or CANTP.

Generating the A2L information

For generating the A2L information GENy creates automatically the A2L fragments `bsw.a2l` and `bsw_xcp_events.a2l` required for the A2L.

Reference: Information for configuration and detailed instructions are provided in the User Manual AMD.

5.4 Configuration of the XCP Module

Configuration tool GENy

The XCP module is configured with the GENy software component configuration tool. The source code for the XCP slave implementation is then generated based on this configuration.

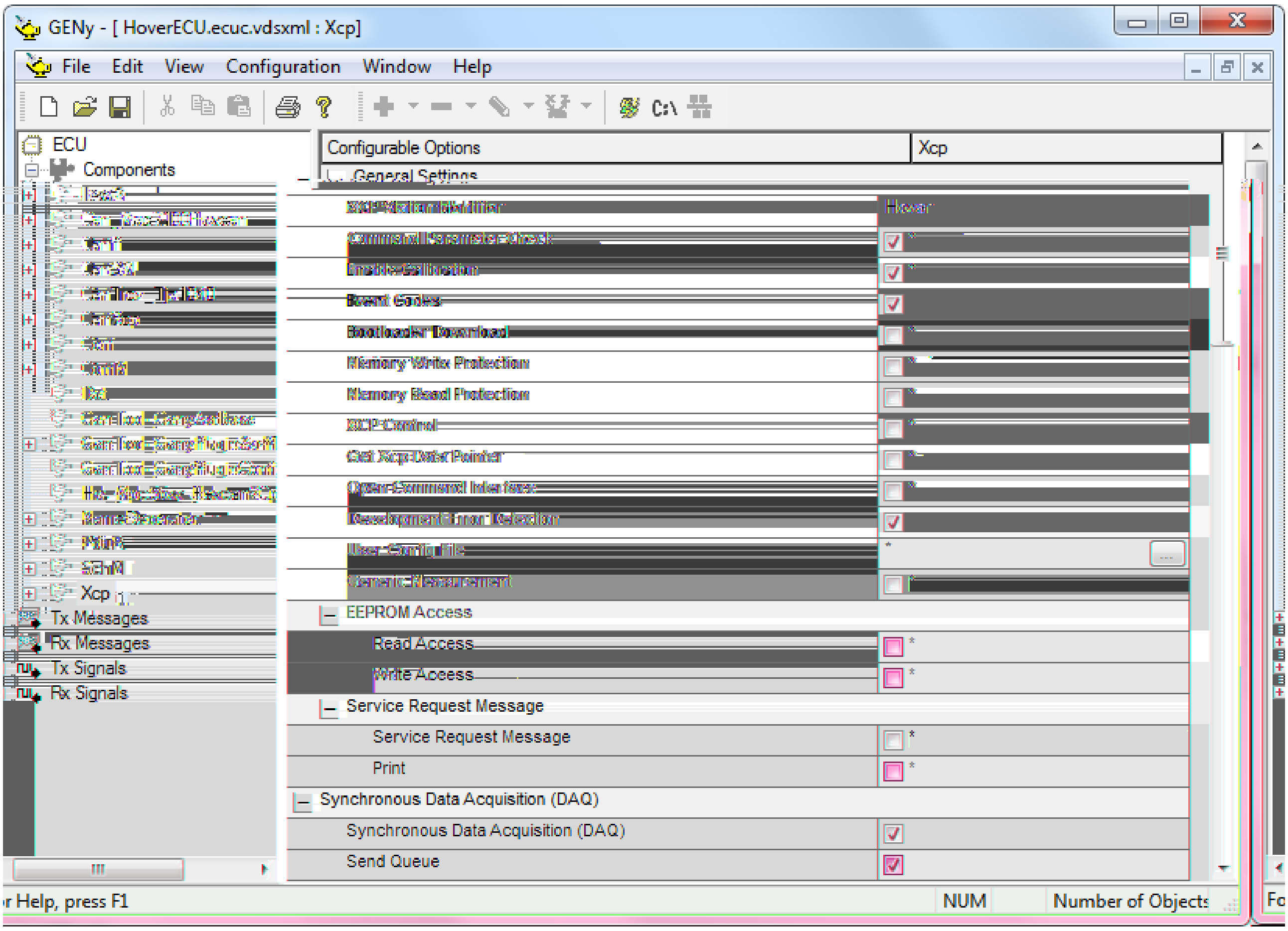


Figure 5-9: Settings in GENy

Reference: Information and instructions on configuring the module can be found in the TechnicalReference_XCP_Protocol_Layer document.

Preface

The most important configuration parameters are described below. In addition, the optional XCP features Measurement Modes and Autoselection and Software Version Check of the A2L File are described in the context of the XCP module.

5.4.1 DAQ List Configuration

Implementation only for dynamic DAQ lists

The XCP module currently has only the implementation for dynamic DAQ lists.

Predefined DAQ lists (static DAQ lists) are currently not supported by the XCP module. Static DAQ lists are not suitable for use of an XCP slave within an AUTOSAR software stack. For one thing, these require an unnecessarily large amount of memory. For another thing, when very many XCP events are implemented, the maximum possible number of static lists may be exceeded if a fixed assignment is used.

Amount of memory space

The amount of memory provided for the DAQ configuration can be specified in the XCP module.

The following formula can be used for the calculation:

$$\text{Memory space for DAQ configuration [bytes]} = 5 \cdot \text{max. number of measurement signals per measure}$$

5.4.2 Tool-Driven DAQ Timestamp Option

Additional options for timestamps

As described previously in the Measurement Modes section, the possibility exists to use a timestamp of the ECU. To do so, this must be supported in the XCP driver. As an additional option, the XCP driver can also supply the timestamp as a matter of principle (timestamp fixed) or on request. The size of the timestamp (1, 2, 4 bytes per event) should be chosen after careful consideration.

Example:
The ECU uses a 1 µs counter for generating the DAQ timestamps. Only a 2-byte timestamp is chosen.

As a result, the timestamp overflows every 65 ms. So that the MCD tool can recognize an overflow, **at least one signal** that supplies a measurement value and thus also a timestamp at a more frequent interval than 65 ms must be measured in the measurement setup.

5.4.3 XCP Event Information

XCP slave to XCP master

XCP event information can be provided in two ways by the XCP slave. Either by a generated A2L file that contains the configured events or by the XCP command `GET_DAQ_EVENT_INFO` which provides the event information directly from the ECU. In both cases the event information has to be configured in the generation tool accordingly.

Caution: If the `GET_DAQ_EVENT_INFO` feature is activated in the XCP module, the automatically generated events of the RTE are not taken into consideration.

Recommendation:

No RTE events are used:

If no RTE events are used, the functionality of the XCP event information can be used. However, attention must be paid that all events that are implemented are described (including those of the BSW component).

RTE events are used:

Due to the fact that the `GET_DAQ_EVENT_INFO` feature overwrites all events defined in A2L files, deactivation of this feature is recommended if RTE events are used. In this case the generated fragment `XCP_events.a2l` can be inserted into the master A2L file (see the Creating an A2L File section).

5.4.4 Software Version Check

Aspects for implementation

The possibilities for checking the software version were previously presented in the Autoselection and Software Version Check of the A2L File section. Aspects for the implementation are explained here.

XCP Station Identifier (protocol command)

The Station Identifier should be centrally defined in an appropriate way and afterwards only integrated. This can be achieved as follows:

- Do not perform a manual configuration of the XCP identifier in GENy.
- Create a "User Defined" configuration containing, for example:

Example:

user_cfg.h:

```
/* Standard commands */
#define kXcpStationIdLength 7u
extern CONST(XcpCharType, XCP_CONST) kXcpStationId[];
```

user_cfg.c:

```
CONST(XcpCharType, XCP_CONST) kXcpStationId[] = "EcuName_V1-2-
0";
```

If this information is integrated in the build process, the Station Identifier `EcuName_V1-2-0` is used.

EPK check

It is recommended that the EPK identifier be generated automatically and consistently with every compilation both in the source code and in the A2L file.

Ideally, the EPK is stored at a constant address in the ECU. This could look like this in the source code:

Example:

```
__attribute__((section("calflash_signature"))) const char
epk[26] = "EcuName V1.2.0 01.03.2012";
```

In the A2L file, the EPK identifier must also be implemented accordingly. For the above example in the ECU software, the entry in the A2L file looks like:

Example:

```
/begin MOD_PAR "EcuName"
       ADDR_EPK 0x350002
      EPK "EcuName V1.2.0 01.03.2012"
/end MOD_PAR
```

Checksum of code segments in the ECU (CANape 11.0 and higher)

So that CANape can calculate the checksum of code segments, some information is required. First, the code segments must be defined in the A2L file. Second, CANape requires a HEX file that also contains the code segments.

5.4.5 Use of the XCP Component in the Implementation

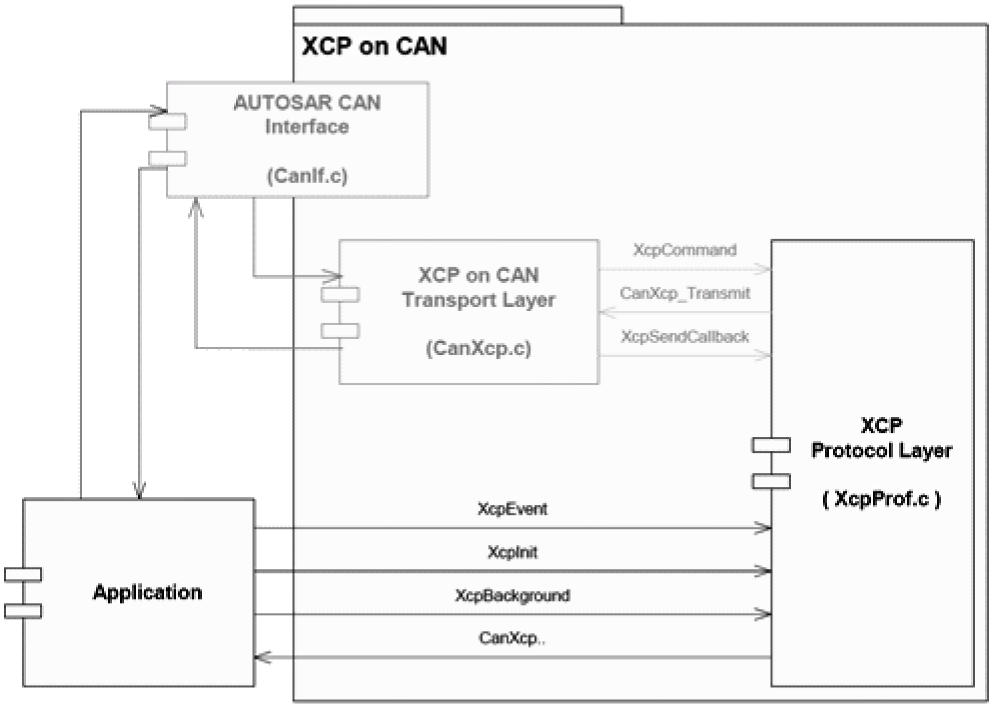


Figure 5-10: Interaction of the XCP module with AUTOSAR application

Interaction of XCP module with AUTOSAR application

1. For DAQ measurements, the basic software or the application calls the `XcpEvent` function.
2. The initialization routine of the application (within `DriverInitTwo`) calls `XcpInit`.
3. The scheduler of the basic software calls `XcpBackground` periodically.
4. By means of the `CanXcp` functions, the application can be informed about CAN-specific events.

Procedure for use

> All modules that require the XCP component must include the `XcpProf.h` header file.

> The XCP component must be initialized in the initialization routine of the software by calling the `XcpInit` function.

> A desired XCP service within the application can be used by calling a function, for example `XcpEvent` (channel) with a corresponding channel/event number.

5.4.6 Recommendations for the Configuration of the XCP Module

Check important parameters

In general, every configuration parameter of the XCP module should be checked with respect to its setting. Important parameters that should be assigned a different value than the default value are described below. These can also be seen again directly in GENy in Figure 5-9.

General Settings

XCP Station Identifier	Manual specification of the file name of the A2L file without the file name extension. Use of the automatic name adaptation described in the Software Version Check section is recommended.
Event Codes	Activate option.
Development Error Detection	Activate option during the development.

Table 5-1: Recommendations for the Configuration of the XCP Module: General Settings

Synchronous Data Acquisition (DAQ)

Synchronous Data Acquisition (DAQ)	Activate option (see the DAQ List Configuration section)
Memory Size	A memory size of 2048 bytes has proven to be adequate. The memory is reserved and used exclusively for the DAQ configuration and the Send Queue for the resume mode.
Prescaler	Activate option.
Write DAQ multiple	Activate option if CAN is not used as Transport Layer.
Resume Mode	Activate option if the OEM requires this in the performance specifications. If activated, the memory size should be rechecked, since the Send Queue should have appropriate capacity.
General Info	Activate option.
STIM	Activate option if the OEM requires the bypassing feature in the performance specifications.
DAQ Timestamp	Activate option (see the Tool-Driven DAQ Timestamp Option section).
Fixed Timestamp	Activate option if CAN is not used as Transport Layer.
Timestamp Size	Selection should be greater than or equal to WORD (2 bytes).
Timestamp Unit + Ticks per Unit	The time unit for the timestamp should be less than the smallest event cycle time.

Table 5-2: Recommendations for the Configuration of the XCP Module: DAQ

Block Transfer

Block Upload	Activate option.
Block Download	Activate option.
MIN_ST for Block Download	Check whether the ECU can process the blocks on time without a loss of data. Otherwise, a wait time should be configured here.

Table 5-3: Recommendations for the Configuration of the XCP Module: Block Transfer

Checksum

Checksum	Activate option.
AUTOSAR CRC Module Support	Recommended if the AUTOSAR module is present.

Table 5-4: Recommendations for the Configuration of the XCP Module: Checksum

5.5 Configuration of the Driver Modules

5.5.1 CAN Module MICROSAR XCP

Configuration

The CAN module MICROSAR XCP is configured with the GENy Software Component Configuration tool.

The CAN messages for the XCP communication can be specified for MICROSAR XCP.

The module is also responsible for creating the `CanXCPAsr.a2l` file.

Reference: Additional information is provided in the Technical Reference XCP Protocol Layer document.

5.6 Configuration of the Memory Management

NVM module

The AUTOSAR Standard provides an NVM module for the memory management. Measuring and calibrating generally have no direct points of contact with the memory management.

The sole use case for configuring the NVM with regard to the XCP module is the use of Resume Mode.

5.6.1 Configuration for Resume Mode

Implementation

For implementation of Resume Mode, the XCP driver must store its DAQ configuration in a non-volatile memory. Two pieces of information must be stored for resume mode: first, the fact that the mode is active and, second, the DAQ configuration itself.

For this a memory block large enough for the configuration is configured in the NVM module. Its size can be derived from the buffer size configured in the XCP module.

Buffer size

The following formula can be used to calculate the buffer size:

$$\text{Buffer size} = \text{Buffer time} \cdot \sum_{i}^{\text{Event}} \frac{1}{\text{cycle time}_i} \cdot \sum_{j}^{\text{Signal}} (\text{Measurement signal}_{\text{Event}i}, j) + \text{Size of the timestamp}$$

API methods

The API methods provided by the NVM module can then be used in order to save and load the configuration in the XCP module. This program part is not generated automatically and must be programmed.

Reference: The methods to be implemented can be referenced in the Technical Reference XCP Protocol Layer document.

5.7 Creating an A2L File

To complete the A2L file, merge all relevant information regarding the ECU

The A2L description file contains all relevant information regarding the ECU. This information is generated from various generators during the creation process. Information, such as the physical address, which is not available until the ECU application has been created, is also needed.

All parts must be merged to ultimately obtain a complete A2L description file. The addresses in the A2L file then still have to be updated.

Ideally, this process is incorporated into the automated creation process of the ECU application.

5.7.1 Creation of a Master A2L File

Note: MICROSAR XCP provides a `_Master.a2l` file as a template in the delivery folder …`\Misc\McData`. All A2L files generated by Vector tools can be found in …`\GenData` folder.

Goal

A master A2L file that merges all partial databases into one is required. This master file can then be used as a template for the file to be created. The objective is to ultimately obtain a single file containing all information.

Project-specific master A2L file

This master A2L file is very project-specific. The information for an A2L file is created by different generators. Some information is also added manually. For this reason, the master A2L file is not created automatically.

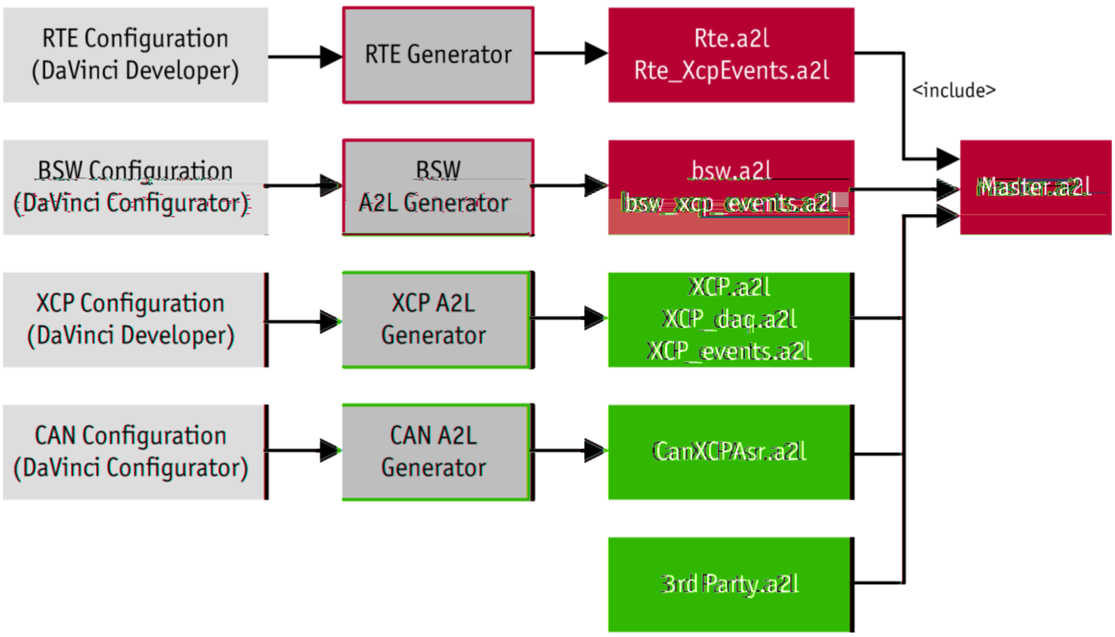


Figure 5-11: Process for creating a master A2L file (example)

`include` commands

The general structure of an A2L file is already described in Figure 3-11: Structure of the A2L file. To merge the individual A2L fragments, `include` commands are used. These are inserted accordingly to the modular structure (AML, General ECU Implementation, IF_DATA and A2L Objects).

To allow the merge of the individual memory segments running smoothly, project-specific adaptions must be made in the master A2L file. These are marked with `// TODO`:

Adaption of `include` commands
For the below-named `include` commands the file paths may have to be adapted. Generally remove the appropriate `includes` if not required in the project.

Use of a text editor
The simplest procedure is to use a text editor to create and adapt the master A2L file.

Master A2L file

AML

General ECU implementation

IF_DATA

```
ASAP2_VERSION 1.60
/begin PROJECT ExampleProject ""
  /begin MODULE MyModuleName ""
    /begin A2ML
      ...
      block "IF_DATA" taggedunion if_data {
        ...
      };
      ...
      //TODO: Include AML Information if required.
    /end A2ML

    /begin MOD_COMMON ""
      // TODO: Set the Byte Order of the ECU as defined by the
      ECUC module MSB_FIRST or MSB_LAST and configure the byte
      alignment used in this project.
    /end MOD_COMMON

    /begin MOD_PAR ""
    /include "GenData\Rte\              "
    // TODO: Add or include MEMORY_SEGMENT information here.
    /end MOD_PAR

    /begin IF_DATA XCP
      /include "GenData\       "
      /begin DAQ
        // TODO: Add or include further a2l file splitter that
        provide XCP Events.
        /include "GenData\           "
        /include "GenData\              "
        /include "Misc\McData\              "
        /include "GenData\Bsw\                 "
        /include "GenData\Rte\                "
       /end DAQ
      /include "GenData\            "
    /end IF_DATA
```

A2L objects

```
    // TODO: Add or include further a2l splitter that provide
    measurement objects.
    /include "Misc\McData\
    /include "GenData\Bsw\       "
    /include "GenData\Rte\       "
    /include "GenData\          "

  /end MODULE
/end PROJECT
```

5.7.2 Expansion of the Master A2L File

`Include` commands

A good approach for incorporating additional contents into the A2L file is the expansion of the master A2L file using `include` commands. Copying additional information directly and inserting it without `include` commands is not recommended.

Integrating of ECU information (General ECU Implementation)

The A2L elements `MOD_COMMON` and `MOD_PAR` are best described in additional A2L files, which are manually integrated in the A2L file via an `include` command.

These `include` instructions are already inserted in the master file and accompany the AUTOSAR Calibration user manual.

Integrating of interface data (Interface Data)

Some parts of the `IF_DATA` information are created by generators. These parts are integrated via an `include` command. If additional manual information is to be added, the creation of additional A2L files is recommended. These must be integrated in the `IF_DATA` at the appropriate points. The merging of `IF_DATA` information from various A2L files using the ASAP2 Merger is not supported.

The `include` instruction `UserDefinedXcpEvents.a2l` in the master file adds manually defined XCP events to the `IF_DATA` section, for example.

Integrating of A2L objects (measurement and calibration parameters)

Partial databases containing measurement and calibration parameters are integrated most commonly. These files can be created, for example, by generators such as Simulink, TargetLink, or the ASAP2 Creator.

Another example is the basic software, which also contains measurable objects.

These files can be integrated manually using an additional `include` command, with the help of the ASAP2 Tool-Set or the ASAP2 Editor.

Note: A file can only be added manually using an `include` command if the file structure permits this. A complete A2L file cannot be added via `include`.

Example: A2L fragment – Inserting via `include` command possible

```
    /CHARACTERISTIC
    …
    /MEASUREMENT
    …
```

Example: Complete A2L file – Inserting possible only via ASAP2 Merger

```
/begin PROJECT ExampleProject ""
  /begin MODULE MyModuleName ""
    /CHARACTERISTIC
    …
    /MEASUREMENT
    …
  /end MODULE
/end PROJECT
```

5.7.3 Working with ASAP2 Tool-Set

5.7.3.1 Merging of Additional A2L Files

Procedure for complete A2L files

A complete A2L file (as in the above example) cannot be embedded in the master A2L file using an `include` command. These types of A2L files can be merged using the ASAP2 Merger program, which is part of the ASAP2 Tool-Set.

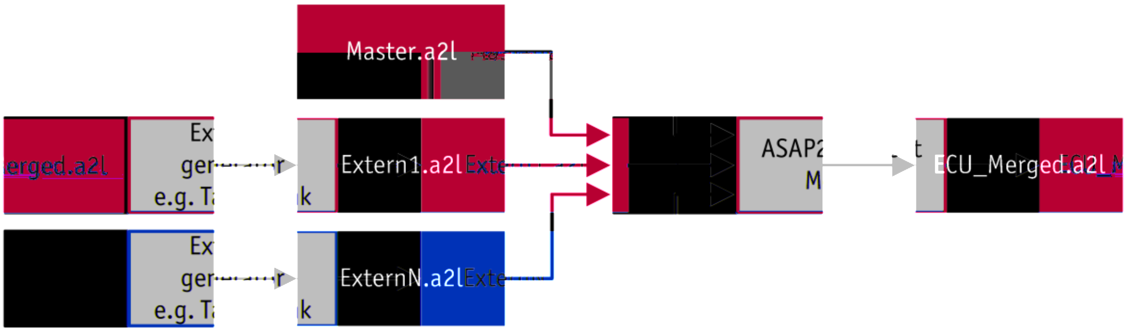


Figure 5-12: Integrating of A2L objects

Reference: The use of the ASAP2 Merger and its possible settings in the INI file are described in the ASAP2 Tool-Set user manual.

Example:
The generated Extern1.a2l and ExternN.a2l files are imported into the master A2L file `Master.a2l` as a slave. The result of the merging is then written to the `ECU_merged.a2l` file. Necessary settings are provided with the `merger.ini` file. The `merger.ini` file must be present since the ASAP2 Merger adopts the setting from this file at each command line call.

Command Line Call:

```
ASAP2Merger.exe -m Master.a2l -s Extern1.a2l -s ExternN.a2l -o
ECU_Merged.a2l -p "<INI_PATH>\merger.ini"
```

Merger.ini

```
[OPTIONS]

MERGE_GROUP_CONTENTS = 1  // The contents of groups with the
                             same name will be merged
```

```
DISABLE_SUFFIXES = 1      // Do not create suffixes for
                             imported objects
```

5.7.3.2 Update of the Addresses in the A2L File

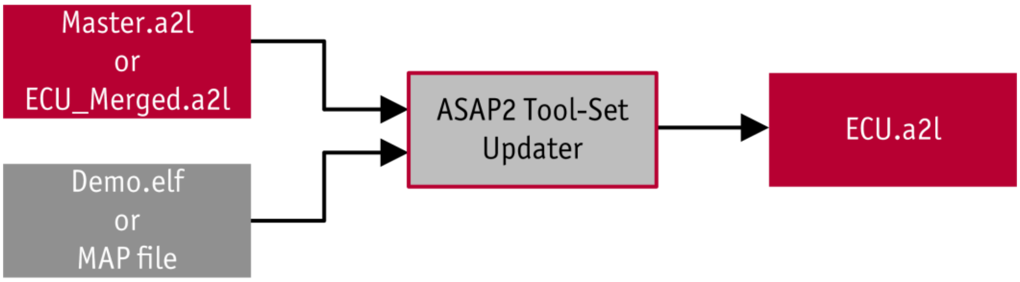
Necessity

It is necessary to update the measurement and calibration parameters in an A2L file because the addresses of objects are not known until after the program code is created (after compilation).

Further benefit

The update step can also be used to create, with the help of the master A2L file, a complete A2L file that no longer has any `Includes`. The advantage of doing this is that afterwards only one file has to be worked with and all partial databases do not always have to be separately copied.

Figure 5-13: Update of the addresses in the A2L file



Reference: The use of the ASAP2 Updater and its possible settings in the INI file are described in the ASAP2 Tool-Set user manual.

Note: The `_Updater.ini` file can be found in the delivery folder …`\Misc\McData`. It is supplied with the AUTOSAR Calibration user manual.

Template `_Updater.ini`

The `_Updater.ini` file is provided as a template, which is indicated by the underscore.

Necessary adaption

The `_Updater.ini` file needs to be adapted in any case, e.g. at least the `MAP_FORMAT` must be specified. The array notation in [ ] is necessary because it is used by MICROSAR that way.

Example:
The `Master.a2l` file is read in and the addresses of the measurement and calibration parameters are updated and written to the `ECU.a2l` file. The addresses for the update are taken from the `demo.elf` file. Information of the update operation is also written in the `a2l.log` file. Necessary settings are provided with the `updater.ini` file. The ASAP2 Updater also always requires an `updater.ini` file.

Command Line Call:
```
ASAP2Updater.exe -I Master.a2l -O ECU.a2l -A demo.elf -L
a2l.log -T "<INI_PATH>\Updater.ini"
```

updater.ini:
```
[OPTIONS]
MAP_FORMAT=31 // Use ELF 32 Bit
```

5.7.3.3 Step by Step Instructions with the ASAP2 Tool-Set

Recommendation

The use of the ASAP2 Tool-Set is recommended because this can be integrated in an automatic generation process. The address update and the export of the database can be integrated as a post-build task.

STEP 1: A2L fragment generation
So that A2L fragments are generated, the corresponding generators must be configured. This is done by integrating these into the build process.

It must be ensured that the created A2L fragments are stored are a fixed location.

STEP 2: Manual creation of A2L fragments
Information that the A2L file must subsequently contain but that is not automatically generated must be manually created.

STEP 3: Adaptation of the master A2L file
A master A2L file must be created. In the process, the paths of the `include` commands must be adapted accordingly.

STEP 4: Merging of additional A2L files
If complete A2L files must be integrated, the Merger of the ASAP2 Tool-Set must be used. For this, the Merger must be called with appropriate parameters for each additional complete A2L file.

STEP 5: Update of the addresses and export of the final file
The final step is to configure the creation of the final A2L file. For this, the ASAP2 Updater is incorporated into the build process, which updates the addresses of the measurement and calibration parameters. At the same time, a new final A2L containing all included A2L fragments is created.

STEP 6: Use of the A2L file in CANape
Following completion of these steps, a current A2L file should now be generated automatically when the ECU software is created.

This final A2L file can then be used in CANape.

5.7.4 Working with CANape and the ASAP2 Editor

Use exported databases without include commands

CANape and the ASAP2 Editor support an interactive procedure for carrying out the actions described above. In this procedure, however, it must be ensured that the master file with its `include` commands remains intact. The master A2L file should therefore not be specified as a database for the ECU directly in CANape. Instead, an exported database that contains no more `include` instructions must always be used.

Caution: When saving, the ASAP2 Editor overwrites the existing A2L file and removes thereby the `includes`. For this reason always store only a copy.

INI-file

All project-specific settings of CANape are stored in the `CANape.ini`. Changes to the configuration can be easily made via the user interface in CANape.

Note: The `_CANape.ini` file can be found in the delivery folder …`\Misc\McData`. It is supplied with the AUTOSAR Calibration user manual.

Template

The _CANape.ini file is provided as a template, which is indicated by the underscore. The necessary presettings, such as for the export important notation [ ] of arrays is already preconfigured to facilitate the implementation.

5.7.4.1 Step by Step Instruction

STEP 1: A2L fragment generation
So that A2L fragments are generated, the corresponding generators must be configured. This is done by integrating these into the build process.

It must be ensured that the created A2L fragments are stored are a fixed location.

STEP 2: Manual creation of A2L fragments
Information that the A2L file must subsequently contain but that is not automatically generated must be manually created.

STEP 3: Insert INI file
Copy the definite `CANape.ini` file to the directory of the master A2L file.

STEP 4: Adaptation of the master A2L file
A master A2L file must be created. In the process, the paths of the `include` commands must be adapted accordingly.

STEP 5: Start the ASAP2 Editor
Start the ASAP2 Editor and load the master A2L. The ASAP2 Editor will be used to create the final A2L file.

STEP 6: Merging of additional A2L files
The ASAP2 Editor can merge content from existing A2L databases. If complete, A2L files must be integrated; the import functionality can be used. Either use **File | Import** or **File | Add partial database** from the application menu.

STEP 7: Update of the addresses
The address update requires a configured MAP file. A MAP file can be added via the database properties. After assigning a MAP file, the address can be updated via the application menu **File | Update addresses**.

STEP 8: Create final A2L file to use in CANape
The master A2L file should not be altered with the ASAP2 Editor. A new A2L file should be generated instead. This can be achieved by saving into a new database using the application menu entry **File|Save as**.

This final A2L file can then be used in CANape.

5.8 Fast Access to the ECU Via the VX Module

Great measurement bandwidth

An VX module is a scalable solution with maximum performance for measurement and calibration tasks. The use of VX measurement hardware enables a greater measurement bandwidth. The system forms the interface between the ECU and a measurement and calibration tool such as CANape. For a high data throughput with minimum runtime effects on the ECU, the data access occurs via microcontroller-specific data trace and debug interfaces. The VX module is connected to the PC using XCP on Ethernet. The VX measurement hardware is connected to the ECU via a POD (Plug-On Device).

Application notes

For information on general integration of a VX module (VX1000), refer to the following application notes:

> AN-IMC-1-016 VX1000: Getting Started with Nexus JTAG and MPC5554
> AN-IMC-1-013 VX1000: Getting Started with Infineon XC2000
> AN-IMC-1-014 VX1000: Getting Started with Infineon TriCore

Note: These documents are available from the Vector Download Center.

5.9 Additional Topics

Items to consider

The following items require additional consideration:

> Memory protection unit, ISO26262, Thread safety
> Limiting of runtime of a task or runnable
> MultiThreading

6 Delivery Test/Quick Start

Objectives

This chapter describes a delivery test for the A2L file created by the supplier. However, it can also be used as a CANape Quick Start for the OEM.

Test of the A2L file

To ensure the completeness and the functionality of the delivered A2L file, a simple delivery test can be performed with the help of CANape. If the A2L file is incomplete or corrupt, an error appears when the file is inserted. If the insertion is successful, a few measurement signals can be added to display windows for the test and a measurement started. If no error appears, the A2L file is functional.

Perform delivery test (step by step instruction):

1. Copy the A2L file to an empty directory and connect the hardware.
2. Start CANape from this directory (right-click on **canape32.exe | Properties | Run in** insert directory of the A2L file).
3. Use a drag & drop operation to move the A2L file to CANape.

   If an error message appears, the A2L file is incomplete or corrupt. Otherwise, the ECU is shown as online.
4. Open the Symbol Explorer and expand the database under **Devices**.
5. Select individual measurement and calibration parameters, use drag & drop to move them onto the empty display page (see Figure 6-1), and choose suitable measurement and calibration windows.

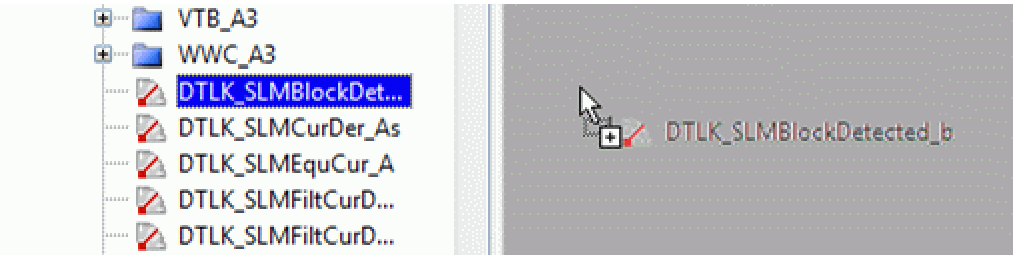


Figure 6-1: Dragging measurement and calibration parameters onto display page

6. Start the measurement and calibrate the calibration parameters.
7. Check the required XCP features in the corresponding settings (for more detailed information on each feature, refer to the CANape online help or the XCP Features in CANape section).

   The delivery test is successful if no error message occurs, meaningful measurement values are displayed in the display windows, calibration parameters can be calibrated, and all desired XCP features can be found.

7 CANape Introduction

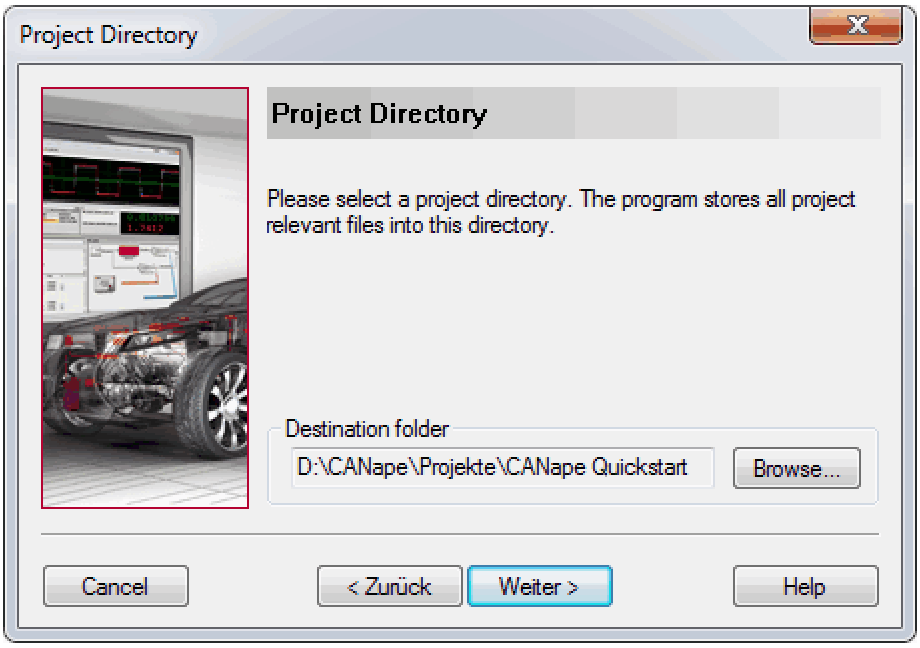
In This Chapter You Will Find the Following Information:

7.1 Creation of a Project

First steps

A CANape project is created either using the selection dialog after starting CANape or in CANape itself via **File | New project**. Once a project name has been defined in the first step, CANape suggests a project directory structure in the second step, in which the project name is a subdirectory.

Figure 7-1: Creating the project directory



Working directory

This serves as a working directory for CANape and should be changed as required. It typically contains the following:

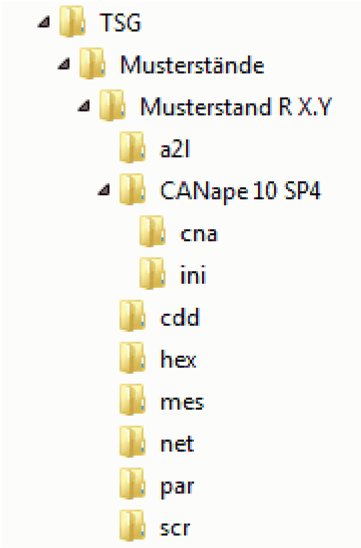
> The `CANape.ini` initialization file, i.e., the global configuration of the project
> Several configuration files (`*.cna`), i.e., local configurations for individual measurement and calibration tasks
> A subdirectory in which the measurement files are stored
> For each ECU:
  > A subdirectory containing the A2L file
  > A subdirectory in which its parameter set files are stored
  > Other subdirectories, depending on the devices used (e.g., external measurement equipment modules)

Definition of the devices

After the desired project directory structure has been specified, the new project is opened. The next step is to define the devices. An ECU description file in A2L format or a diagnostic driver in ODX/CDD format is generally required for this. In the end, a complete project has at least one configuration file (`*.cna`), the corresponding initialization file (`*.ini`), and at least one ECU description file (`*.a2l`).

Figure 7-2 shows the recommended project directory structure.

Figure 7-2: Project directory



Prototype version release

Folders for the project-relevant files are created for each prototype version release X.Y of an ECU. The CANape configuration file (`*.cna`) and the `canape.ini` file are located in folders in the **CANape 10 SP4** subdirectory. The Hex file, the databases (`*.a2l`, `*.cdd`), and the network files (e.g., `*.axml`) are inserted as subfolders for each prototype version release. In addition, the measurement, parameter set, and script files are stored in their own folders.

7.2 Device Configuration

Settings

The settings for **devices**, **networks**, and **channels** can be modified and individual devices and networks can be added in the device configuration. The device configuration is accessed via the icon or using **Device | Device configuration**.

Graphic representation

The device configuration can also be represented graphically using the Device window. Double-clicking the individual icons opens the corresponding part of the Device Configuration or the Database Editor.

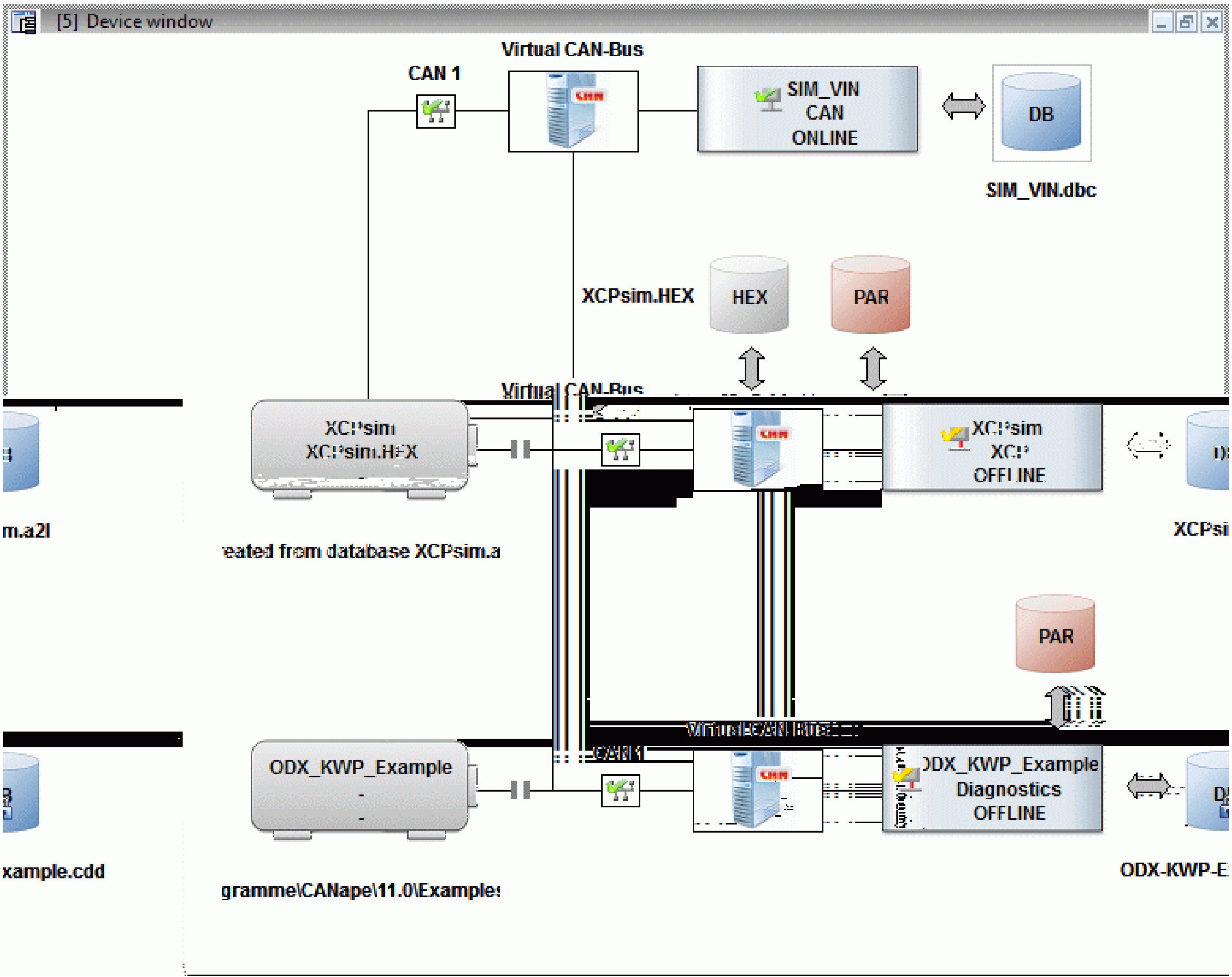


Figure 7-3: Device window in CANape 11.0

7.2.1 Devices

Creating new devices

The **Devices** subitem of the Device Configuration displays all the created devices. Here, new ECUs can be created from a database or the MCD3 server, or completely new ECUs can be created. In the latter case, CANape generates an A2L body that the user must still configure and complete using the ASAP2 Editor. Besides the XCP and CCP devices, diagnostic drivers or databases can also be used. An example of integrating a diagnostic database and of using panels for this can found in the installation directory of CANape under **Examples | ODX**. A new device can be created directly by dragging and dropping the database in CANape.

Bus monitoring

For the bus monitoring, the databases of the CAN bus (`*.dbc`), FlexRay bus (`*.fibex`), and LIN bus (`*.ldf`) can be integrated in CANape. In the AUTOSAR context, the possibility exists to use an AUTOSAR system description file (`*.arxml`) in the case of the CAN or FlexRay bus.

Configuration

Corresponding dialog pages are available for configuring each created device. Additional information regarding the configuration options can be found in the CANape online help. Depending on the device status, the icon changes from green (online) to yellow (offline) or red (inactive).

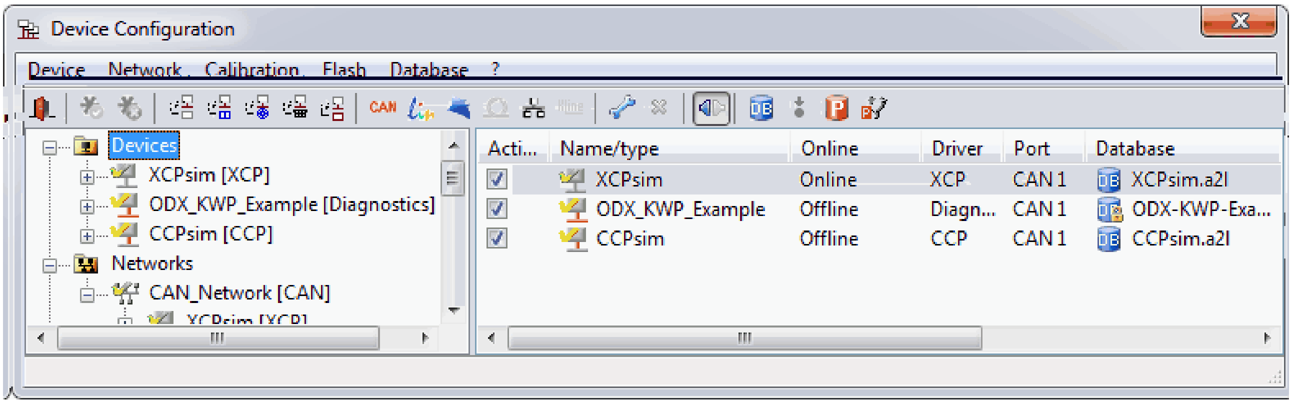


Figure 7-4: Device configuration

7.2.2 Networks

Listing

The **Networks** subitem lists all networks available in the current configuration.

Configuration

The following networks can be created in CANape: CAN, LIN, ETH, K-Line, FlexRay, and MOST. The networks are configured on the corresponding dialog pages.

7.2.3 Vector Hardware

Configuration of the hardware

The configuration of the hardware is performed using the Vector Hardware. It can be opened using **Device | Vector hardware configuration** or in the **Channels | Vector** section in the Device Configuration.

The appropriate hardware can be assigned to the respective channels using **Application | CANape**. In so doing, the physical channel number does not have to match the logical channel number. The possibility also exists to change the number of channels for a particular bus system.

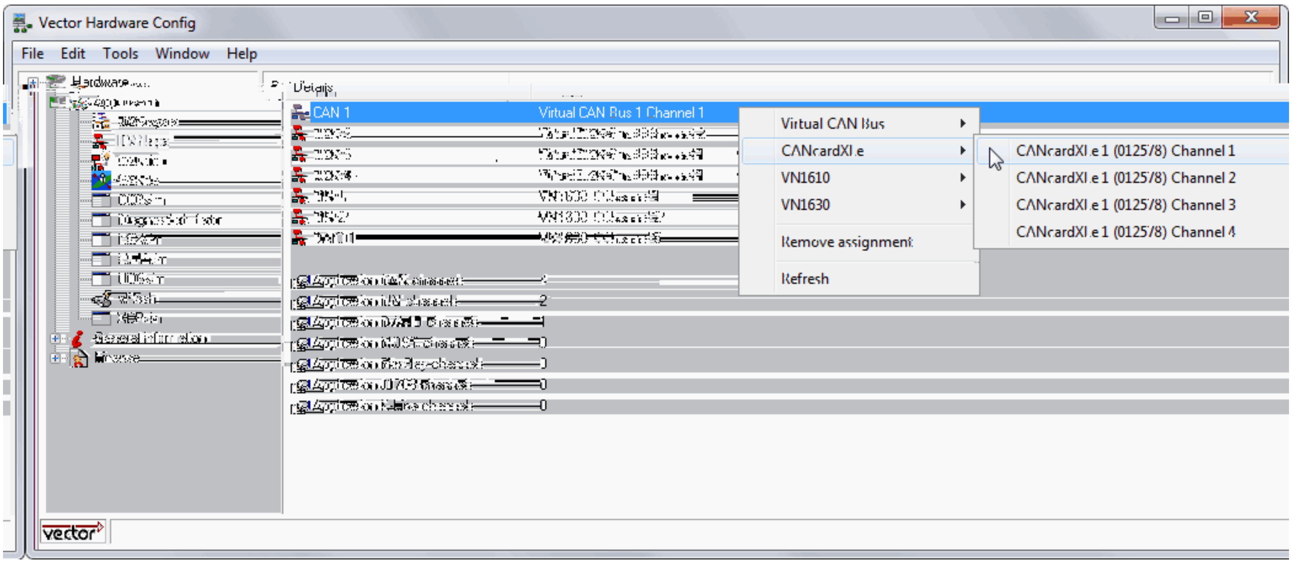


Figure 7-5: Vector Hardware Config

7.2.4 XCP Features in CANape

Timestamp

The use of a timestamp can be specified in the Device Configuration in subitem **Protocol | Event List** of the device. Depending on the implementation in the ECU, the option also exists here to require a timestamp of the slave.

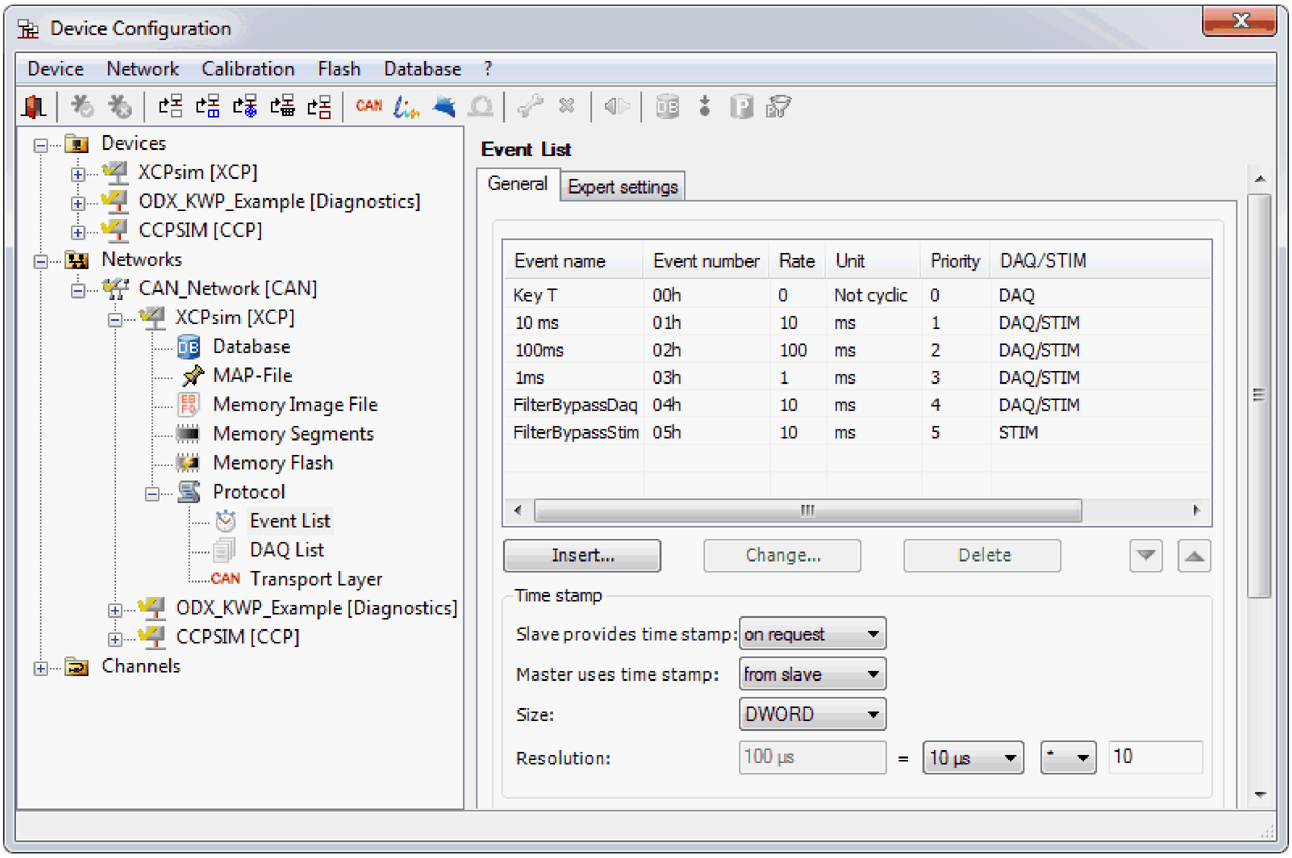


Figure 7-6: Timestamp in the device configuration

Resume mode

Whether or not resume mode is supported is indicated in the **Expert settings** of the **DAQ Lists** subitem.

Autoselection/ software version check

The autoselection and the software version check of the A2L file can also be set in the device configuration. This option can be found in the **Database** subitem.

If the "Page Switching" or "Checksum calculation" options are used, these can be found under Memory Segments of the device (see Figure 7-7).

Online help

All XCP features are described in more detail in the CANape online help.

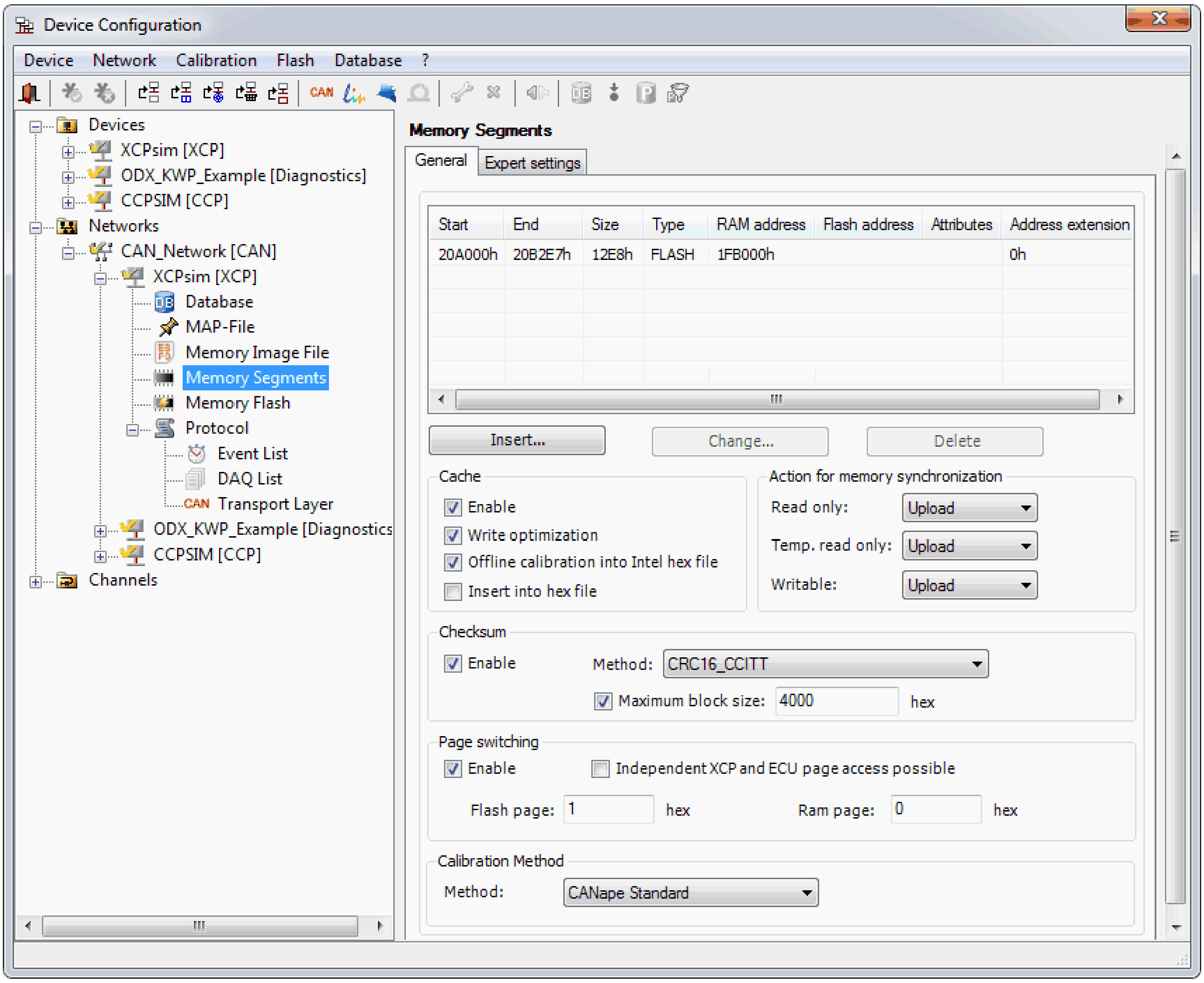


Figure 7-7: Page switching in the device window

7.3 Online Measurement Configuration

Call The complete measurement is configured in the online measurement configuration. It is called via the icon or using **Measurement | Measurement Configuration**.

Display windows and pages Various display windows are available in order to display the measurement and calibration parameters. These windows are described in detail in the CANape online help. In addition, several display pages can be created to enable a well-organized complete configuration.

7.3.1 Measurement Options

Behavior of the measurement The behavior of the measurement can be configured in the measurement options of the measurement configuration. For example, the handling with polling signals during the measurement or the size of the measurement buffer can be adapted. In addition, a comment template for newly created measurement files can be specified here.

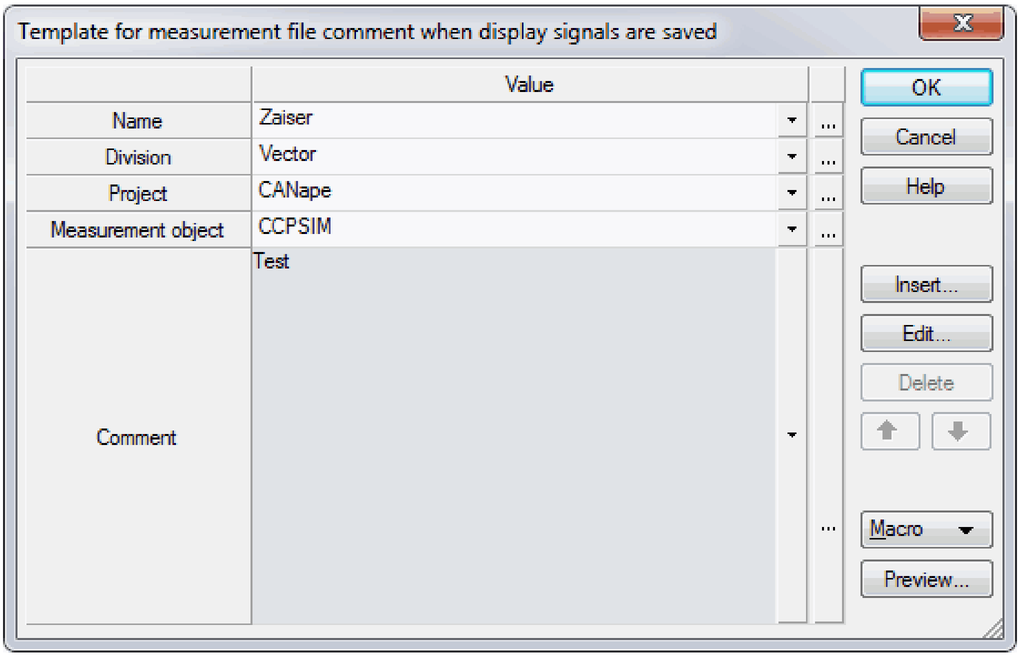


Figure 7-8: MDF measurement comment template

7.3.2 Measurement Signals

Measureable signals

All signals of the measurement configuration are listed on this page. Signals of the database can be selected using **Edit | Insert Signal**. Only the signals that are contained in the measurement signal list or in the display windows of CANape are measured. The option also exists to deactivate signals for individual measurements instead of deleting and adding them again. For the case that a signal is to be measured but not recorded, i.e., for performance and memory space reasons, the **Recorder** option can be deactivated.

Measurement modes

The measurement mode of the measurement signals leaves some of the configuration options up to the user. The most commonly used measurement modes are:

> **Event:** In event mode, the ECU sends the current measurement value of a signal autonomously. The possible events and DAQ lists are defined in the ECU and described in the A2L file.

> **Polling:** In polling mode, the measurement values of a signal are returned asynchronously on request and according to the polling rate of the ECU. This process is well suited for slower measurements when there are no requirements for synchronous polling.

> **Cyclic:** In XCP and CCP, the cyclic measurement mode corresponds to the event measurement mode. A data reduction can be achieved based on its cycle time.

> **On key:** When a key (combination) is entered, the signal is requested (polling).

> **On trigger:** When a trigger event occurs (StartTrigger, StopTrigger, LastTriggerFinished), CANape measures the desired signal (polling).

> **On event:** When a particular system event occurs (e.g., measurement start), the signal is measured (polling).

Measurement rate

The measurement rate is displayed to the right of the displayed measurement mode in the measurement configuration of the measurement signals. It indicates the recording rate in polling or cyclic mode. The rate is specified as a time interval between two measurement values, in milliseconds.

Bus utilization

The bus utilization and the measurement events for the selected device are listed at the bottom of the measurement configuration. Here the bar indicates the percentage utilization of the individual event time bases and the bus.

Online help

In addition to the signals of the individual databases, additional measurement signals such as global variables can also be incorporated into the measurement configuration. For more detailed information on this, refer to the CANape online help.

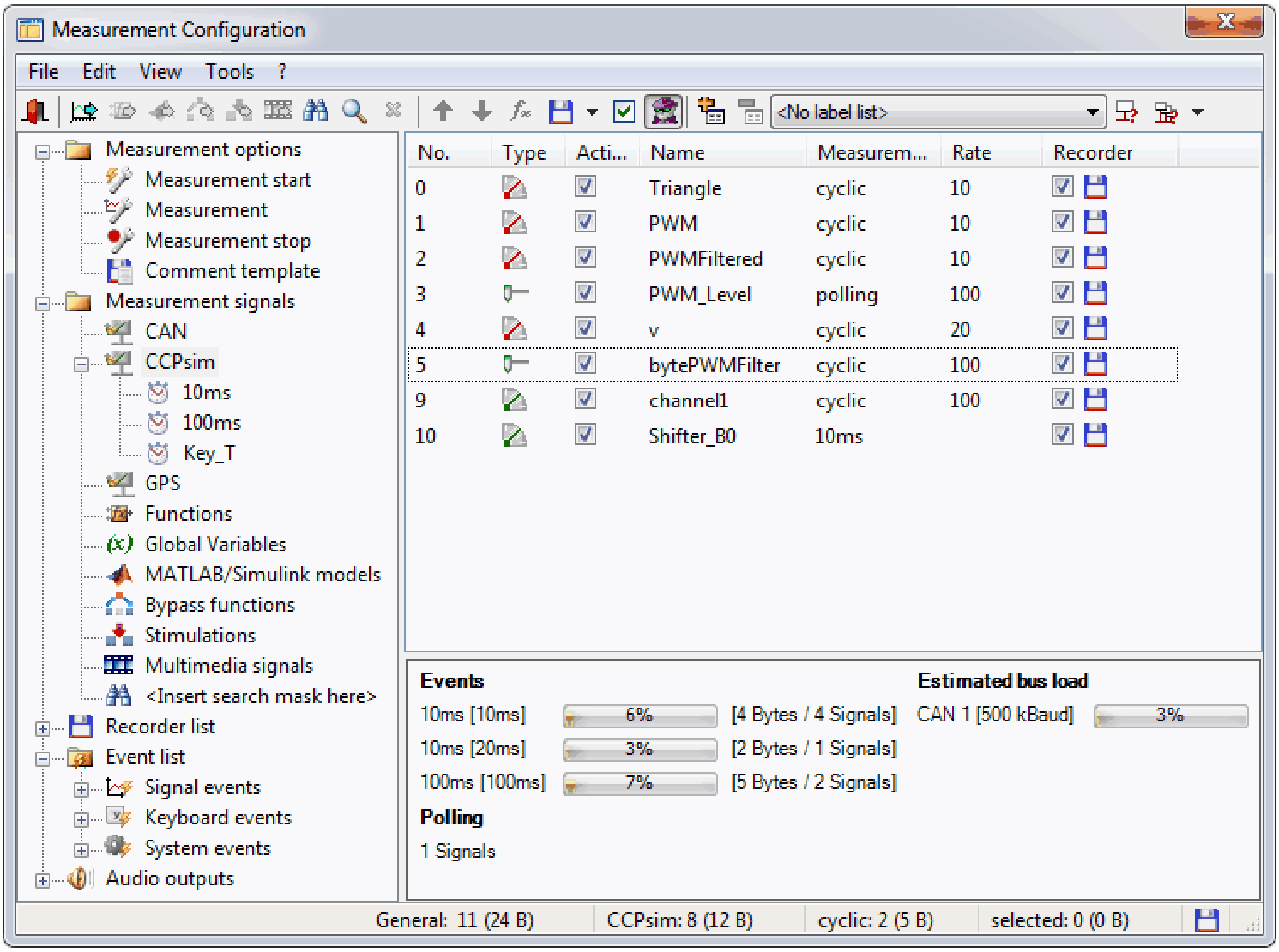


Figure 7-9: Measurement configuration: Measurement signals list

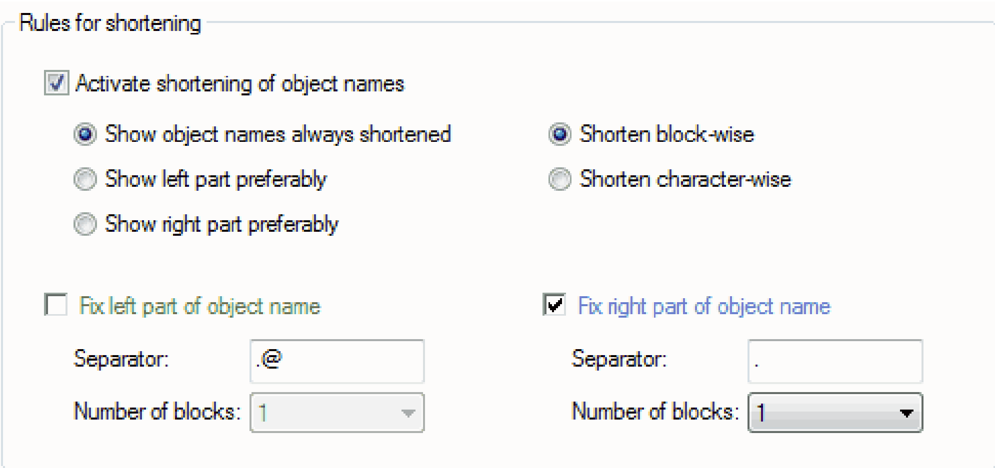
Inserting signals

With the help of the Symbol Explorer , individual measurement signals can be inserted directly in a display window using a drag & drop operation. These are automatically added to the measurement signal list.

Shortening rule

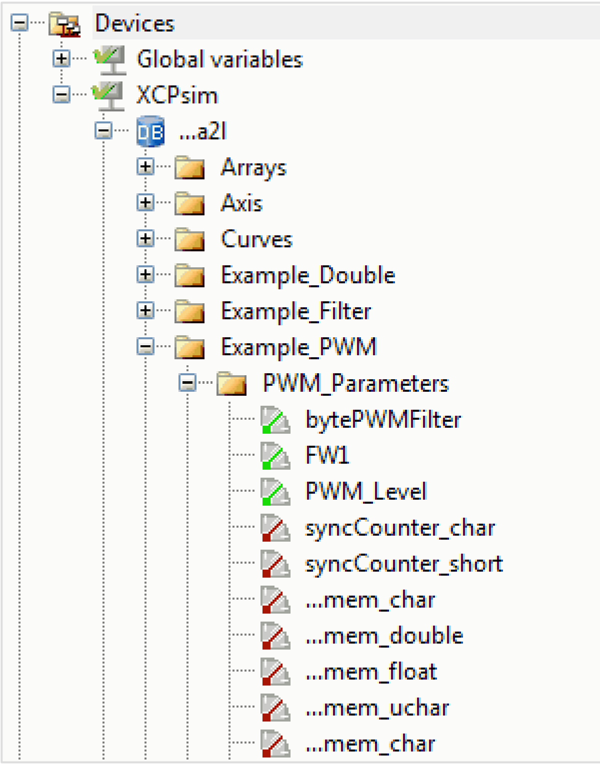
To improve the readability of long measurement signal names in the Symbol Explorer, a shortening rule can be specified using **Tools | Options**, section **Display | Object Names**.

Figure 7-10: Setting of a name shortening rule



This indicates the start of the signal name only and is limited in the display to the last part after the specified separator.

Figure 7-11: Example for use of a name shortening rule



7.3.3 Recorder List

Definitions/Settings

The recorder list in the measurement configuration provides an overview of the defined recorders. The option exists to deactivate individual recorders in order to realize different measurement tasks. The setting of the file name of the MDF file can be made individually for each recorder. Here, it is possible to use different macros in order, for example, to record the time of day in the file name. Under the **Options** area, various settings can be made for each recorder. A detailed explanation of these settings can be found in the CANape online help.

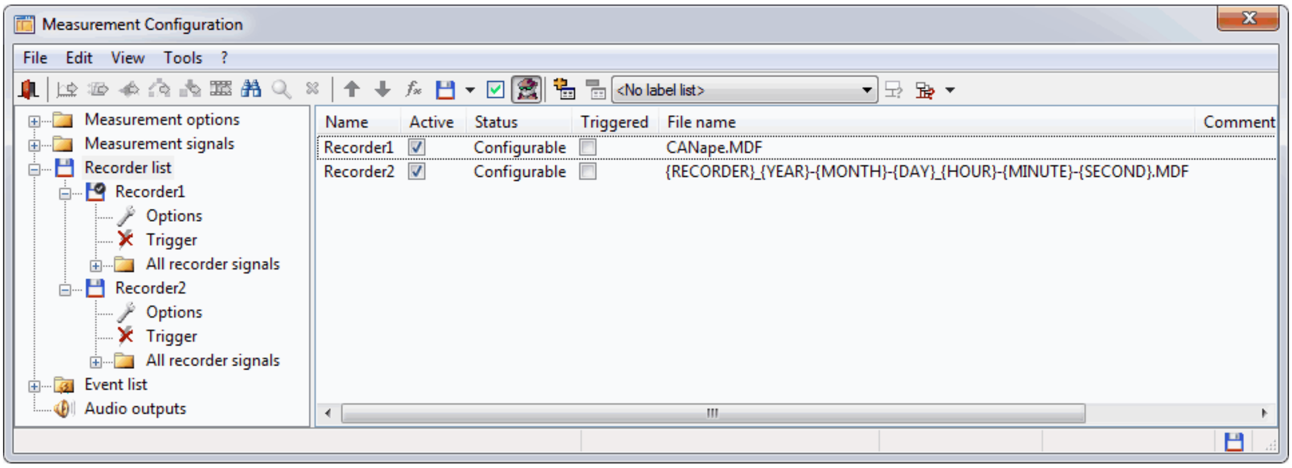


Figure 7-12: Measurement configuration: Recorder list

Trigger of recordings

In addition to the most straightforward measurement in which all signals are recorded over the entire measurement period, the possibility also exists to trigger the recording of individual signals by certain events. These are defined in more detail in the **Trigger** area.

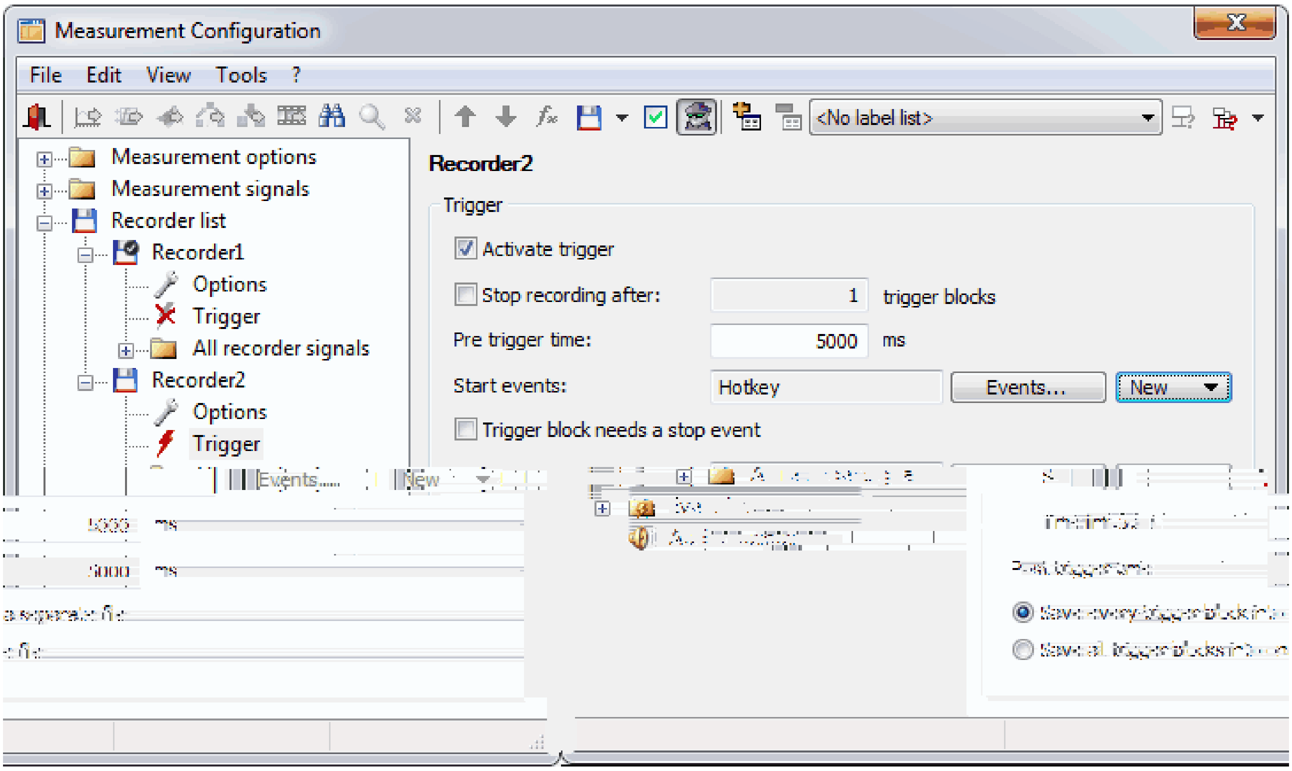


Figure 7-13: Trigger condition

Start events

The selection menu of the **[New]** button can be used to select various start events. The following categories are available for selection here:

- System events (messages from the PC or the ECU)
- Signal events (values from active measurement)
- Keyboard events (operator inputs)

Stop event

These events are also available again as a stop event. However, a time limitation can also be chosen as a stop event.

Assign signals to recorders

The measurement signals that are recorded by this recorder are indicated under **All recorder signals**. Signals can be assigned to individual recorders so that these are recorded only when the trigger condition occurs.

7.3.4 Event List

Overall event list

The **Event list** section of the Measurement Configuration lists all events with their properties. Here it can be seen whether the event is an ECU event or a general system event. The defined trigger events are also displayed here.

Definition of new events

New events are defined using the context menu. These are then available in the measurement signal list as a measurement mode so that, for example, a signal is measured only after a particular key has been pressed.

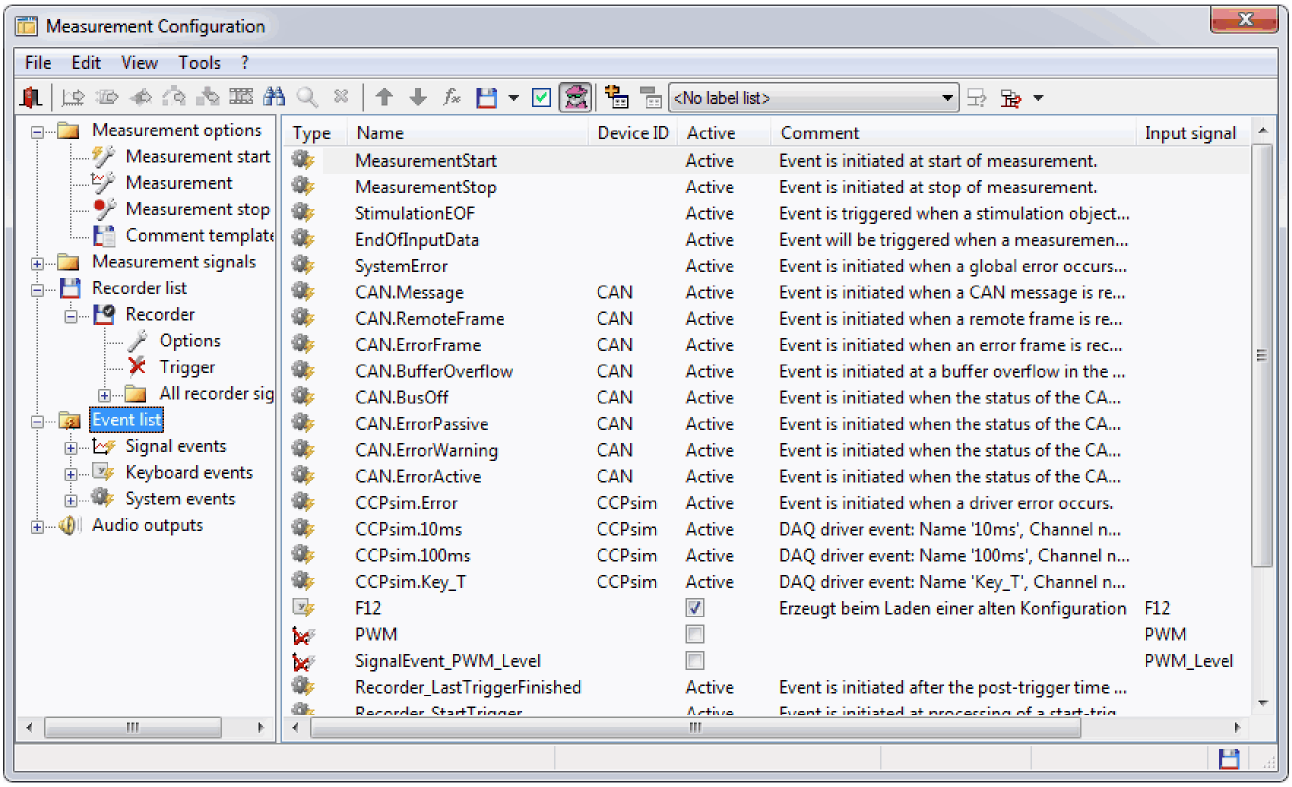


Figure 7-14: Measurement configuration - Event list

7.4 Working with Parameter Set Files

Purposes for saving parameter set files

CANape offers the option to perform online calibration of calibration parameters and to save these as a parameter set file. These files are then used mainly for two purposes:

> For saving the current version and for documenting and/or exchanging parameter values

Different options are available for selection for saving the calibration parameters. First, the parameters of a single calibration window can be saved by selecting **Save** in the popup menu of the Calibration window. Second, all the parameters of all Calibration windows can also be saved. This can be done using **Calibration | Save all calibration windows**. In addition, the possibility exists to select particular parameters via a filter (**Calibration | Save parameter set as**).

- In order to bring the system to a defined state

  Several functions are also available for loading a parameter set file. Calibration parameters in a particular calibration window can also be opened here by selecting **Load** in the popup menu of the Calibration window. Particular calibration parameters can also be selected using **Calibration | Load parameter set from**.

7.5 Dataset Management

Definition of dataset

A dataset is a set of various parameters at a particular point in time within the edit history. It normally contains all parameters that belong to an ECU and is represented via the following files:

- Database file (A2L file)
- Memory image content (HEX file)
- Parameter set file (only for datasets from the eCDM system)

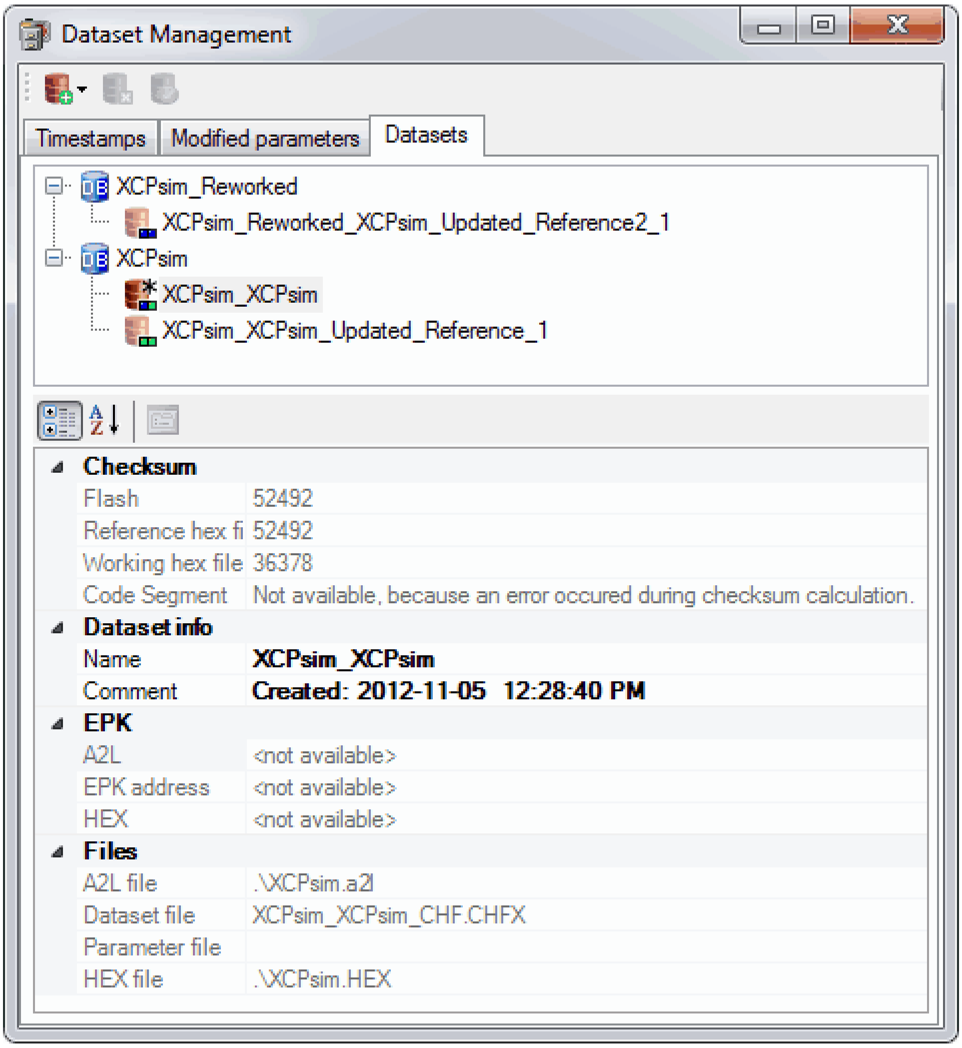
The dataset is the central object for the versioning and configuring of parameters.

7.5.1 Tool-Based in CANape 11.0 and Higher

Dataset management

In CANape 11.0, a convenient dataset management tool has been introduced. The **[Dataset Management]** can be called via the device configuration. Here, various datasets of an ECU can be added. New datasets (A2L+HEX, HEX or uncoded) can be added on the Datasets tab using the icon. Additional settings are available in the context menu. The **Timestamps** tab shows the snapshots of the calibration history and indicates their timestamp.

Figure 7-15: Dataset management in CANape 11.0



Working with multiple datasets

The datasets are then displayed and can be activated in the Symbol Explorer. This provides a convenient means for working with multiple datasets within a project.

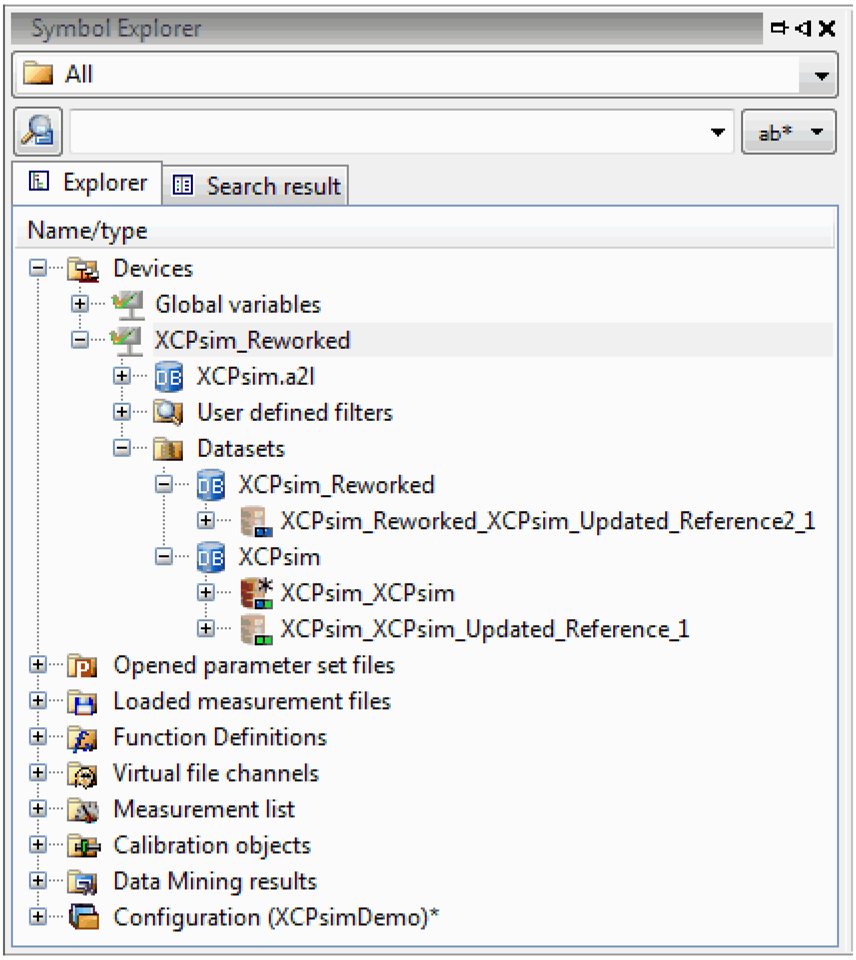


Figure 7-16: Dataset management in the Symbol Explorer

Demo project

The **Examples** folder of the CANape installation directory contains a demo project named `Datasets_Thesaurus`, which illustrates the use of the dataset management using an example.

7.6 Offline Evaluation

Read in measurement data

For purposes of offline evaluation, measurement data can be read in using **Analysis | Show values from measurement file**.

Measurement File Manager

The **Measurement File Manager** (can also be opened via the **Analysis** menu item) shows all loaded MDF files as well as the virtual MDF channels. Several possible settings are available in the toolbar of the Measurement File Manager. These are described in detail in the CANape online help.

Data Mining

An automatic procedure for offline evaluation of loaded MDF files is available under **Analysis | Data Mining**. The option exists, for example, to find the times at which the speed exceed 3000 rpm. In so doing, it is possible to evaluate multiple measurement files with measurement signal names as identical as possible in a single search. These are specified in the **File filter list** section. The option of using wildcards (`*.mdf`) is also available.

Calculation methods

The calculation methods are configured in the **Methods** section. The following are available for selection here:

> Function (based on user-defined functions that are created in the function editor or from the global function library)

> MATLAB/Simulink model (based on MATLAB/Simulink models that are available as DLL)

> Arithmetic condition (based on user-defined criteria)

> Script (defined in the Functions Editor)

Definition of algebraic conditions

Figure 7-17 shows the definition of an algebraic condition. The time range to be evaluated can be set under **Extended**.

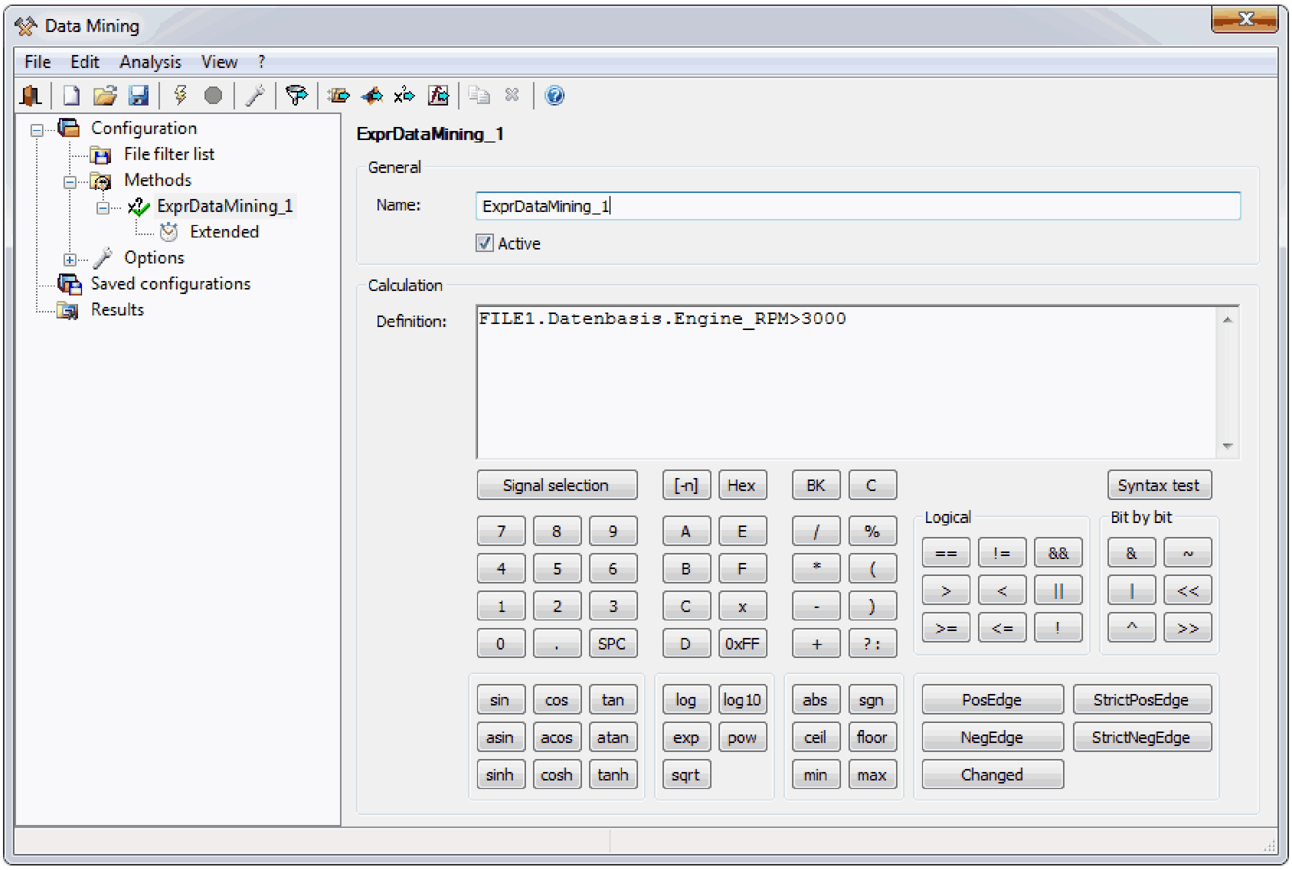


Figure 7-17: Data Mining: Creating an algebraic condition

Naming analysis files

The desired file name of the analysis file can be entered in the **Options** section. The name can contain various macros that can be inserted using the corresponding button.

Further settings

In addition, it is possible to limit the number of hits per file. It is useful to specify a creation date of the file to be searched if only the measurement data starting from a certain date are to be evaluated.

Output in CSV format

The results can also be output in CSV format for further analysis. The desired separator for the measurement data should be indicated in the selection menu in this section.

Executing scripts	Scripts that are executed before starting the analysis, before the analysis of each file, after the analysis of each file, or after finishing the complete analysis can also be specified.
Example of Data Mining	A detailed example of Data Mining can be found in the installation directory of CANape under **Examples | DataMining**.

7.7 Flashing

Flash tools	Other Flash tools, such as vFlash can be opened from CANape.
Online help	Additional information on the topic of flashing with CANape can be found in the CANape online help.

8 Addresses

Addresses on Vector homepage

Please find the contacts of Vector Informatik GmbH and all subsidiaries worldwide via:

http://www.vector.com/vi_addresses_en.html

9 Abbreviations

Abbreviation	Description
ASAM	**A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems
AUTOSAR	**AUT**omotive **O**pen **S**ystem **AR**chitecture
BSW	**B**asic **s**oft**w**are
CSA	**C**ommon **S**oftware **A**rchitecture
CTO	**C**ommand **T**ransfer **O**bject
DAQ	**D**ata **A**cquisition
DTO	**C**ommand **T**ransfer **O**bject
E/E Architecture	**E**lectrical/electronic architecture
EPK	**EP**ROM-**K**ennung (EPROM identifier)
EPROM	**E**rasable **P**rogrammable **R**ead **O**nly **M**emory
MCD System	**M**easurement **C**alibration, and **D**iagnostics System
ODT	**O**bject **D**escription **T**able
RTE	**R**untime **E**nvironment
SW-C	**S**oft**w**are **C**omponent
VFB	**V**irtual **F**unction **B**us

Get more Information!

Visit our Website for:

> News

> Products

> Demo Software

> Support

> Training Classes

> Addresses

www.vector.com