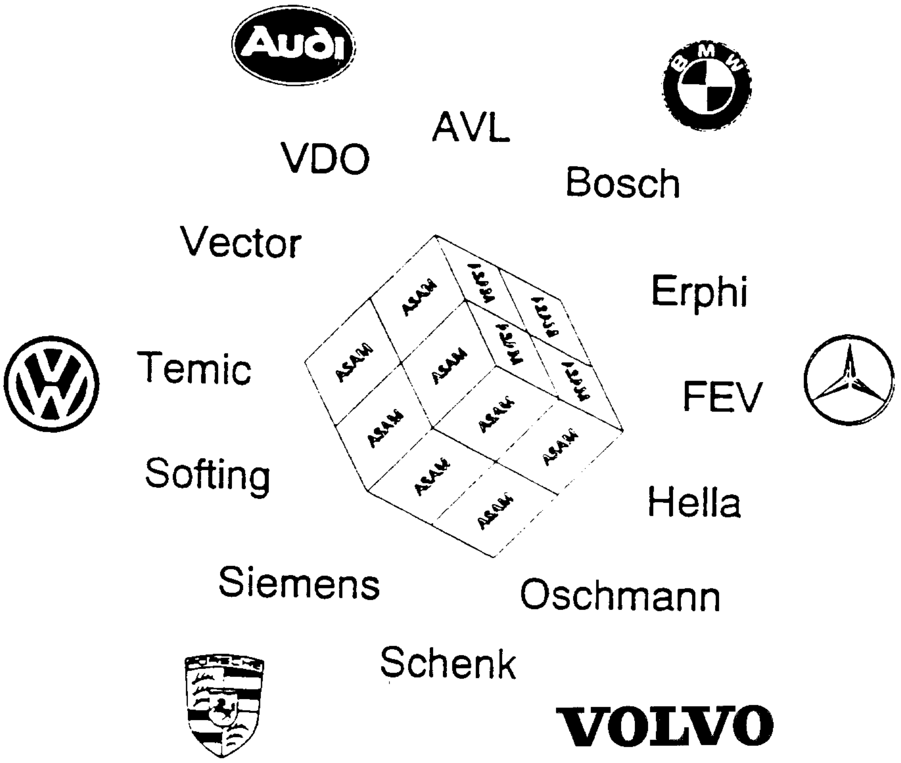


ASAP

Applications Systems Standardization Working Group

Interface Specifications

Interface 2

Version 1.31 of 06/15/1999

ASAP2: STANDARDIZED

DESCRIPTION DATA

Authors:	Dr. H. Schelling	Fa. Vector Informatik
	Dipl.-Ing. P.Lampert	Fa. Vector Informatik
	Dr. C. Dallmayr	Fa. Softing
	Dipl.-Inf. G.Luderböck	Fa. Softing
	Dipl.-Ing. M.Eisele	Fa. Vector Informatik

Version:	1.31
Date:	06/15/1999

Contents

1 Introduction

This document is based on specification "Application Systems Standardization Working Group: Interface Specifications - Interface 2, Version 1.0 of 31 March 1994". The new requirements result form experience applying this standard. The modifications are decided at the meeting of ASAP2 Working Group at 24 January 1996. Appendix D contains a list of modifications relating to Version 1.0. At the meeting of ASAP-plenum on 11 March 1997 the acceptance of the present first draft of this specification had been decided.

At ASAP2-Plenum on 24 June 1997 a final revision was decided to correct some details (version 1.21: no functional extensions).

2 ASAP: Goals, Method, Interfaces

The working group for the standardization of application systems (ASAP) was created in the autumn of 1991 at the initiative of the development boards of the German automobile manufacturers.The motivation is cost reduction based on co-operation in non-product relevant fields. The initiative is supported by the automobile manufacturers Audi, BMW, Mercedes-Benz, Porsche and VW together with the contractors active in this segment Bosch, Hella, Siemens, Temic, VDO and free suppliers and automation suppliers such as AVL, Erphi, FEV, Schenk, Softing and Vector.

The working group for the standardization of application systems (ASAP) aims at making the tools and methods generated during the development phase of vehicle electronics compatible with each other and hence interchangeable.

To this end the system parts required for the application as well as for verification and testing are adapted to the current, technical requirements in parallel with the design and development phase of the actual control units and thus brought to a high maturity level. In practice these efforts make it possible to incorporate individual components of the overall system into the process chain and to integrate them with other application environments.

To reach these goals, the ASAP group has agreed to subdivide the overall system into subcomponents (Figure 1) using commonly defined interfaces that are compatible and interchangeable.

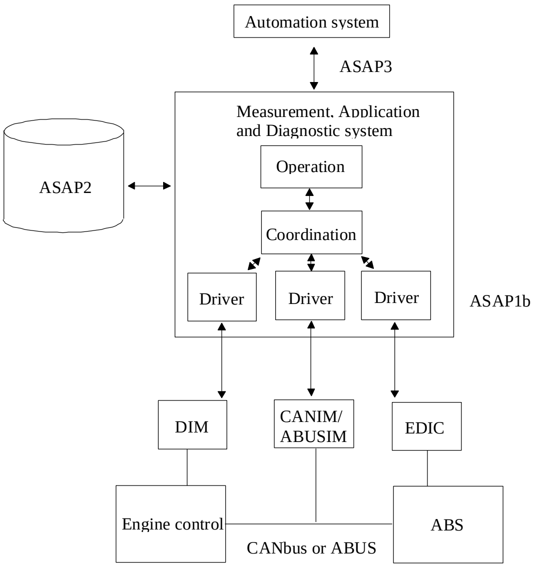


Figure 1: Overall system and interfaces

The individual application systems (AS) of the measurement, application and diagnostic system (MCD) are linked to the automation via interface ASAP3, they obtain information about the control unit's internal elements, its interfaces and communication methods from the ASAP2 description file, and are in turn linked to the control units (ECU) and the control unit dependent measurement technology (ADC) via the ASAP1 interface via ROM emulators, CAN or ABUS or the diagnostic bus (D bus). This structure allows current monolithic applications to be divided into compatible subsystems.

Via the ASAP1b interface the standard connection of the control units and of the control unit dependent measurement technology is realised independently of the chosen communication path or the relevant supplier of the control unit. To obtain this functionality, the control unit or the measurement system is linked to the measurement, application and diagnostic system via the transport path, the interface hardware and a driver in accordance with the ASAP specifications. This subsystem below interface 1b is identified by the ASAP device (Figure 2):

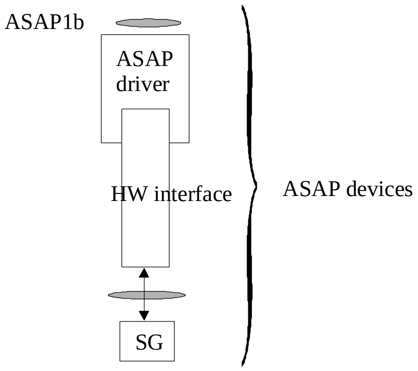


Figure 2: ASAP device (principle structure)

An ASAP device thus defined may be composed of different elements depending on the selected connection :

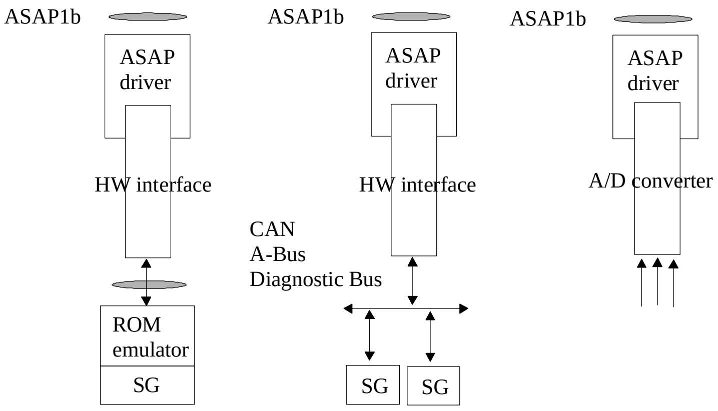


Figure 3: Various versions of the ASAP device

The ASAP 1b interface is a functional interface which was initially defined independently of the MCD operating system. It offers a range of services controlling the exchange of parameters and data via the ASAP 1b. These ASAP services are:

Service	Name	Function description
1	INIT_READ	Initialisation of the measurement system
2	INIT_ACCESS	Initialisation of the adjustment system
3	SYNC	Synchronisation of the individual subsystems
4	READ	Data transfer upstream of ASAP 1b
5	ACCESS	Data transfer downstream of ASAP 1b
6	STOP	End of measurement data collection for intelligent module type 2..4
7	FREE_HANDLES	Enable references - reject subsystem measurement data
8	GIVE_STATUS	Explicit status interrogation in the event of an error
9	COMMAND	Handling of manufacturer defined commands

Table 1: ASAP Services Interface 1b

This allows similar subsystems such as ROM emulators of the firms AB and XY to be accessed in the same way. This 'similarity' is related to the use of the above-mentioned ASAP services which imply above all commonly agreed communication procedures. Special features and different communication methods within the ASAP device are embedded in this device, the setting of parameters occurs by allocating equally special binary objects and parameters from the ECU description file. Interpretation of the objects is only possible by the ASAP device. Furthermore, uniform communication with the control unit is possible independently of the selected connection. Different features of the various control unit connections can be balanced within the ASAP device, while the functional communication, e.g. the setting of parameters, is also standardised.

The ASAP description file (ASAP2) is used to describe the ECU internal data. This ASAP subsystem can be created from a number of different subelements:

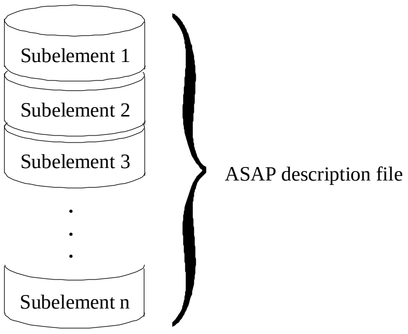


Figure 4: ASAP description file

The subelements of the ASAP description file are :

- project relevant information,
- data structure in the control unit,
- external interfaces,
- communication methods as ASAP device, and
- the conversion procedures for representation in physical units.

An ASAP description file constitutes the reference for an individual control unit and its link to the ASAP interface 1b. It contains the following information :

Subelement	Contents	Examples
1	project-related data	Name of the relevant user of the subcontractor, data reference number
2	ECU-internal structures	Status and structure of the warm-up characteristic map, content of the user information field
3	Conversion rules	Scaling values for conversion from hexadecimal to physical for subelements 2, 4 and 7
4	Measurement channels	Addresses, resolution and update rates of the measurable RAM cells
5	Methods	Parameters of ASAP device for communication setup with the ECU, e.g. hexcode for emulator
6	HW layout	Segmentation of the related memory modules in the ECU e.g. data block size
7	SW interfaces	Description of the communication contents on the CAN bus or ABUS, e.g. identifier and data content of the messages

Table 2: Subelements of ASAP interface 2

The individual subelements differ in terms of content, supplier and person functionally responsible as well as in terms of customer and his application system. However, the same information storage method is used throughout to allow for the global management of the individual components.

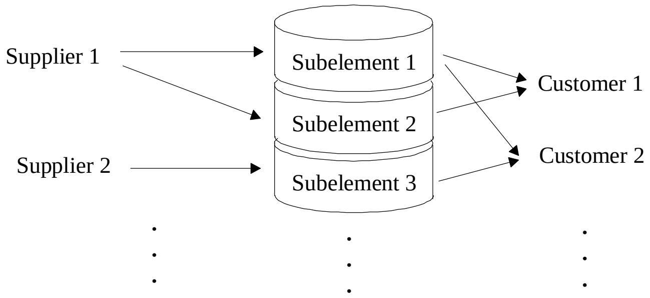


Figure 5: Customers and suppliers of the ASAP 2 subelements

This makes it possible to replace independent subelements e.g. in the case of a functional extension of the CAN interface: subelement 7, SW interfaces, section CAN, as well as subelement 1, project-related data, and thus to generate different description files corresponding to the current state of the control unit, which may then serve as input for other application systems.

Interface ASAP3 links up the measurement, application and diagnostic system (MCD) to an automation system (AuSy). From the standpoint of the AuSy it can therefore be considered as an intelligent device as e.g. an indexing device or a fuel scale. It incorporates both the various methods for access to control units as well as the individual structures in the control units and offers the following higher-level functions:

- starting an application on the MCD
- executing the diagnostic function
- executing the application function
- executing the measurement system function

These functions are offered by applications with an ASAP3 interface (e.g. give current engine speed, give content of warm-up characteristic map, give error memory) without the AuSy having to know the control unit specific details. The MCD offers its services to the AuSy at a quasi higher level. These services are based on the information on addresses, scalings, methods etc. in control units, interfaces and ASAP devices, which is obtained from the ASAP2 description file:

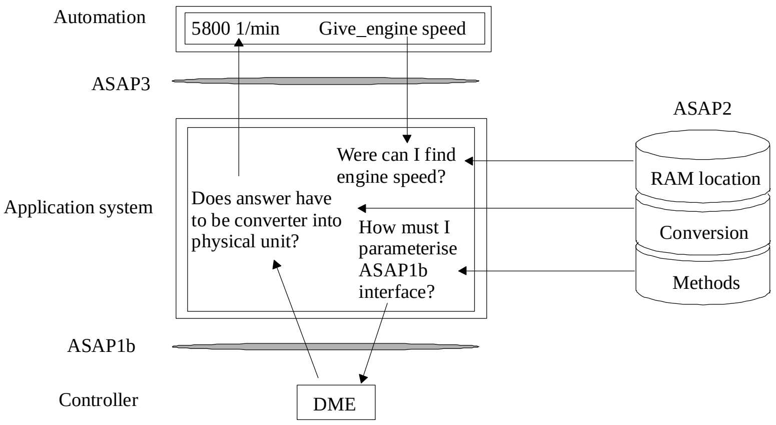

Automation
5800 1/min
Give_engine speed
ASAP3
ASAP2
Application system
Were can I find engine speed?
Does answer have to be converter into physical unit?
How must I parameterise ASAP1b interface?
RAM location
Conversion
Methods
ASAP1b
Controller
DME

Figure 6: Interworking of Automation system, Application system, DME with ASAP interfaces

The AuSy does not require any special knowledge about the methodology and the parameter setting of the interfaces to the control units for its automation sequences. This service is provided by the MCD. The connection is established via the ASAP 3 interface.

PART A: DIVISION OF THE DESCRIPTION DATA

3 Division

The definition of the ASAP 2 interface and hence the specification of the ASAP2 data base is aimed at defining a database independently of a computer or an operating system in such a way that a transparent and manufacturer-independent standard is established.

From the application point of view the database in accordance with the ASAP 2 interface contains the complete description of all control unit relevant data in a project. A project consists of project specific header data and one or more control unit specific descriptions. These control unit descriptions (= description of an ASAP device) include all conversion formulas and explanations about the applicable (adjustable) and measurable (non-adjustable) quantities and present a format description of the interface specific parameters (for ASAP interface 1b). The measurement, application and diagnostic system need only evaluate the quantities (and their conversion etc.), but not the interface specific parameters. The latter are only passed on to the structures of the ASAP 1b driver. To make sure that these structures are correctly filled the MCD must know the parameter type. The type is communicated with the ASAP2 metalanguage (see part C).

A project may include the control unit descriptions of various control units from different suppliers. The descriptions differ in terms of content, but use a common information storage methodology to allow for a global management of the project components. An INCLUDE mechanism allows to summarize the various control unit descriptions of various projects (Single-Source-Concept).

The ASAP2 database thus consists of a number of different subcomponents structured in accordance with the following diagram. The MODULE keyword denotes an independent ASAP device.

```
PROJECT
    {
    HEADER{...}                          /* Project description */

    MODULE ASAP_DEVICE1
        {
        MOD_PAR{...}                     /* Control unit management data */
        MOD_COMMON {...}                 /* Module-wide (ECU specific) definitions */

        CHARACTERISTIC{...}              /* Adjustable objects */
        CHARACTERISTIC{...}
        ...
        MEASUREMENT{...}                 /* Measurement objects */
        MEASUREMENT{...}
        ...
        COMPU_METHOD{...}                /* Conversion method */
        COMPU_METHOD{...}
        ...
        COMPU_TAB{...}                   /* Conversion tables */
        COMPU_TAB{...}
        FUNCTION{...}                /* Function allocations */
        FUNCTION{...}
        ...
        RECORD_LAYOUT{...}           /* Record layouts of adjustable objects */
        RECORD_LAYOUT{...}
        }
    MODULE ASAP_DEVICE2
        {
        MOD_PAR{...}                 /* Control unit management data */
        MOD_COMMON {...}             /* Module-wide (ECU specific) definitions */

        CHARACTERISTIC{...}          /*Adjustable objects*/
        CHARACTERISTIC{...}
        ...
        MEASUREMENT{...}             /* Measurement objects */
        MEASUREMENT{...}
        ...
        COMPU_METHOD{...}            /* Conversion method */
        COMPU_METHOD{...}
        ...
        COMPU_TAB{...}               /* Conversion tables */
        COMPU_TAB{...}
        ...
        FUNCTION{...}                /* Function allocations */
        FUNCTION{...}
        ...
        RECORD_LAYOUT{...}           /* Record layouts of adjustable objects */
        RECORD_LAYOUT{...}
        }
    MODULE ASAP DEVICE 3
        {
        ...
        }
    }                                /* END OF PROJECT */
```

The keywords defined in the ASAP2 database are described in Part B.

4 System Description (SG Verbund)

Within the scope of the standardization of application systems, it is intended to simultaneously apply control units of various manufacturers (e.g. engine control/ transmission control/anti-slip control). The ASAP2 database supports this: under one project header, which describes the overall system, the control unit descriptions (ASAP devices) of a number of manufacturers can be included, e.g. by 'INCLUDE'. Thus, an efficient overall standardization can be implemented with a common application system.

Figure 7 gives a schematic representation of the ASAP devices to be commonly applied, summarized under one project

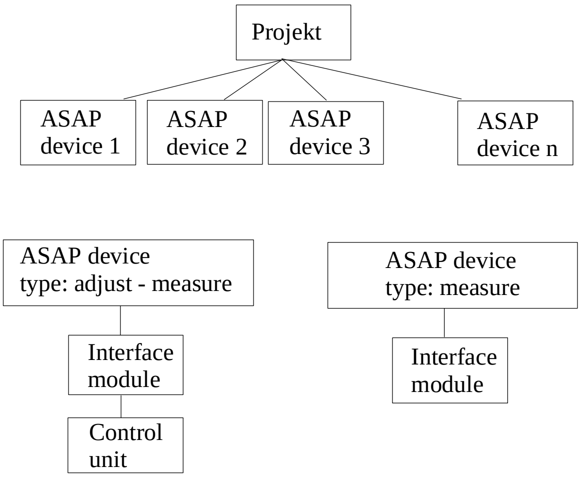


Figure 7: Model application system

An ASAP device can thus be a control unit with related interface and ASAP 1b driver or only an interface with driver. The second case applies if e.g. quantities, exchanged via the bus as message, are monitored via ABUS or CAN (bus monitor).

The project header (keyword HEADER, page 109) is typically created by the project manager and includes the following entries:

- version of the parameter file format [keyword VERSION, page 197]
- project
- project identifier
- project number [keyword PROJECT_NO, page 156]
- remarks, comments

5 Description Application Devices

For each ASAP device a complete description (keyword MODULE, page 136) is created. The description is supplied by the relevant supplier of the control unit. The components of this description are:

- control unit management data (e.g. users responsible,...)[MOD_PAR, page 143]
- control unit internal structures (e.g. standard record layout) [MOD_COMMON, page 141]
- communication interfaces [IF_DATA, see Part C]
- adjustable and measurement objects [CHARACTERISTIC, page 58 and MEASUREMENT, page 131]
- conversion rules [COMPU_METHOD, page 67]

5.1 Control unit management data

The control unit management data are created with the MOD_PAR (page 143) keyword. Using optional parameters the following can be specified:

– data status	[VERSION, page 197]
– comments, remarks	
– EPROM identifier address	[ADDR_EPK, page 30]
– EPROM identifier	[EPK, page 87]
– Manufacturer or supplier	[SUPPLIER, page 180]
– Firm or customer	[CUSTOMER, page 76]
– Customer number	[CUSTOMER_NO, page 77]
– User	[USER, page 184]
– Telephone numbers (applications engineer responsible)	[PHONE_NO, page 154]
– Control unit	[ECU, page 86]
– Processor	[CPU_TYPE, page 73]
– Number of interfaces	[NO_OF_INTERFACES, page 149]
– Memory layout	[MEMORY_LAYOUT, page 134]
– System constants	[SYSTEM_CONSTANT, page 182]

- Project-basis-address (see MEMORY_LAYOUT: memory layout)

5.2 General description data (control unit internal structures)

These description data (keyword MOD_COMMON, page 141) make it possible to specify a number of parameters for the module as a whole (i.e. for this ASAP device). The following optional parameters can be specified:

– Standard record layout (is there one standard layout?)	[S_REC_LAYOUT, page 183]
– Standard deposit mode of the axis points (difference, absolute values)	[DEPOSIT, page 80]
– Byte order	[BYTE_ORDER, page 56]
– Data size in bits	[DATA_SIZE, page 78]

5.3 Interface Parameters (general parameters)

A format description of the interface specific parameters must be made available by the supplier in ASAP2 metalanguage (A2ML: see Part C). The measurement, application, diagnostic system need not know the driver parameter settings: to read the interface specific parameters only a description of the data types is required.

5.3.1 Interface module (memory emulator)

These parameters describe the access methods to the measurement data collection:

- Display table type (code patch in the ECU for measurement data output), address display table(s), maximum length of the display table(s)
- Output: which memory address
- Triggering: trigger segment address

5.3.2 CAN bus

If in a specific project a number of application devices have access to the CAN bus, a consistency check must be carried out for these paramteres in the conversion program (physically only one CAN bus, all corresponding data records of the 'interface parameter' type must match).

- Bus timing
- Configuration
- Bus parameters
- Access methods fo measurement data collection:
  - Collection through passive hearing or remote frames
  - Note: Here there is no description of the CAN identifier
- possible description of a protocol for data adjustment

5.3.3 ABUS

These parameters describe the communication via the ABUS (VW):

- Bus operating mode (reference, difference)
- Bus parameters (number of scannings, error figures etc.)
- ADEX functions realised in the drivers

5.3.4 Bus parameters for serial protocols (ISO)

Here parameters describing serial communication are stored:

- Protocol identification (which serial protocol)
- Protocol parameters (timing), protocol specific
  - byte interval of tester
  - byte time-out of control unit
  - lock sequence time of tester
  - block time-out of control unit
  - timeout during communication setup
  - entry time during free-running

- ECU address
- Access methods
  - telegram answer
  - telegram-answer-answer ... (free-running)
  - display table (free-running)
    - length and structure of the display table
- Communication setup process (if necessary according to the protocol)
  - type of trigger
    - none
    - 5 Baud
    - low level
    - fast trigger (10.4 kBaud)
  - send telegram for initialisation (dummy)

- source type
  - time synchronous
  - crankshaft synchronous
  - event synchronous
  - asynchronous

5.3.5 Analog interface

Does not require any interface parameters!

5.4 Adjustment objects (one description per adjustment object)

Each adjustment object must have its own description to be created with the CHARACTERISTIC keyword (see page 58). Between the /begin CHARACTERISTIC and /end CHARACTERSITIC brackets further keywords can be nested (as may also be the case for other keywords).

Example:

```
/begin CHARACETRISTIC PUMPKF          /* Description of a characteristic map */
        ...
        ...
        /begin AXIS_DESCR             /* axis description for x axis */
                ...                   /* ...nesting depth 2 */
                ...
                MAX_GRAD   7.0        /* gradient limited, nesting depth 3 */
        /end AXIS_DESCR
        /begin AXIS_DESCR             /* axis description for Y axis */
                ...
                ...
        /end AXIS_DESCR

/end CHARACTERISTIC
```

The description of the adjusment objects contains references to (possibly) common conversion methods for a number of adjustment objects, record layouts or functions. The referenced objects are described only once under their own keyword.

5.4.1 Deposit structure [CHARACTERISTIC]

Mandatory parameters:

- Name of the adjustable object
- Comment, function description
- Type of adjustable object
  - value, value block, curve, map, ASCII, pointer
  - fixed curve, fixed map
  - group curve, group map, axis point distribution,
  - vector curve, vector map
- address, address location, addressing type

- record layout (enumeration or reference)
- maximum increment for incrementing or decrementing a function value

Optional parameters:

- Axis description [AXIS_DESCR, page 45]
  - reference to input quantities
  - fixed curve or field: parameters for calculation of the axis point values
- Maximum gradient of the adjustable object between two neighbouring axis points (Delta_W/Delta_St)
- Monotony (with reference to an axis)

Remark:

The orientation (deposit: in rows or columns) as well as the word length of the axis points and function values (8 bit or 16 bit) is given in the layout.

5.4.2 Bit pattern conversion (axis points and function values)

- Reference to the list of conversion methods
- Byte order [BYTE_ORDER, page 56], signed-unsigned information
- Plausibilities (limit values)
- Physical representation: see Conversion method

5.4.3 Function orientation (Reference)[FUNCTION_LIST, page 108]

- List of those 'functions' that are allocated to this object. The referencing occurs by names.

Remark:

This solution does not correspond with the current method realised in DAMOS (reference occurs inversely with an allocation list in the 'Function Record'.

5.5 Measurement channel (one description per measurement; e.g. AD value, CAN signal, source data, RAM cell)

For each measurement a description is created with the keyword MEASUREMENT (page 131). The description of the interface-specific data must be supplied by the supplier in A2ML.

As for the adjustable objects the description of the measurement object contains a reference to common conversion methods and function descriptions that can be referenced from various measurement objects.

Mandatory parameters:

- name of the measurement channel
- comments, function description
- word length, bit mask for individual bit sizes

5.5.1 Deposit structure (measurement channel)

5.5.1.1 Description of the interface module deposit structure

Here the following can be described (see Part C):

- addresses, address location, address length (e.g. for entry in the display table)

5.5.1.2 Description of the CAN deposit structure

Here the following can be described (see Part C):

- Name of the CAN message
- CAN identifier
- Message length
- Sender
- Signal type (mode signal, mode dependent signal, standard signal)
- Reference to a mode signal (optional: if signal type = = mode dependent signal)
- start bit, signal length

5.5.1.3 Description of the ABUS deposit structure

Here the following can be described (see Part C):

- Name of the ABUS telegram
- ABUS identifier
- Repetition rate of the sender
- Faulty reactions and receiver timeouts

5.5.1.4 Description of the series protocol deposit structure (ISO)

Here the following can be described (see Part C):

- Send telegram (containing full telegram with dummy for address/length)
  - Command byte
  - Command data (detailing the components of 2.3.4)
    - address
    - number of bytes
- Result data
  - position in answer telegram
  - deposit type (byte order, signed-unsigned information)
- Display tables
  - address, address location, address length
  - deposit type (byte order, signed-unsigned information)

5.5.1.5 Analog interface deposit structure

Here the following an be described (see Part C):

- Analog input (No. 1-8)
- Resolution in bits
- Gain, prescaler

5.5.2 Bit pattern conversion

- Reference to conversion method list
- byte order [BYTE_ORDER, page 56], signed-unsigned information

Physical representation: see conversion methods

5.5.3 Function orientation (reference)

Under the keyword FUNCTION_LIST (page 108) a list of the functions allocated to this object is given. Referencing occurs via names (see comment in chapter 6.3.63 page 108).

5.6 Conversion method

The keyword COMPU_METHOD (repeatedly possible, see page 67) creates a list of the conversion methods used during the data adjustment and the collection of measurement data (conversion from the internal format in the emulation memory to the 'physical' representation of a quantity).

Parameters for the conversion method:

- Name of the conversion method
- Comment, function description
- Conversion method type (table with/without interpolation, polynome, verbal conversion table)
- Reference to conversion table [COMPU_TAB_REF, page 71]
- Physical representation (significant positions, decimal places, physical unit)
- Coefficients for fractional rational function [COEFFS, page 65]

Remark:
The 'physical representation' and 'plausibilities' parameters are allocated to the conversion method, as this significantly benefits the size of the description file (empirically there are fewer conversion methods than adjustable objects or measurement objects).

5.7 Conversion tables

Under the keyword COMPU_TAB (page 70) a list is given of the conversion tables used during the data adjustment and the measurement data collection (physical conversion and verbal conversion). It contains the mandatory parameters :

- Name of the conversion table
- Conversion table type (table without/with linear interpolation)
- number of value pairs
- value pairs

For the visualisation of the bit patterns verbal conversion tables (COMPU_VTAB, page 72) can be used (allocation table: bit pattern <--> string)).

5.8 Function description [FUNCTION, page 103]

All functions referenced under CHARACTERISTIC and MEASUREMENT can be described as follows:

– Name of the function
– Comment, description of the *function*

5.9 Record layout [RECORD_LAYOUT, page 160]

Description of the various record layouts of the adjustable objects (see also Appendix B).

PART B: FORMAT OF THE DESCRIPTION FILE

6 FORMAT OF THE DESCRIPTION FILE

6.1 Hierarchic division of the keywords

Keyword	(*)=multiple	Meaning
ASAP2_VERSION		ASAP2 version identification
A2ML_VERSION		Version number of ASAP2 Meta Language
PROJECT		Project description
HEADER		Project header description
PROJECT_NO		Project number
VERSION		Project version number
MODULE	(*)	Description of the ASAP devices
A2ML	(*)	ASAP2-Meta-Language (interface-specific descripition data)
AXIS_PTS	(*)	Axis points distribution
ANNOTATION	(*)	Set of notes
BYTE_ORDER		Byte order of axis points
DEPOSIT		Absolute or difference axis points
DISPLAY_IDENTIFIER		Optional display name
FORMAT		Display format of axis points
FUNCTION_LIST		Function orientation
IF_DATA	(*)	Interface-specific description data
READ_ONLY		'Read Only' attribut
GUARD_RAILS		Indicates the use of guardrails
CHARACTERISTIC	(*)	Adjustable objects
ANNOTATION	(*)	Description
AXIS_DESCR	(*)	Axis description
ANNOTATION	(*)	Set of notes
AXIS_PTS_REF		Reference to axis point distribution
BYTE_ORDER		Byte order of axis points
DEPOSIT		Absolute or difference axis points
FIX_AXIS_PAR		Fixed axis parameters
FIX_AXIS_PAR_DIST		Fixed axis parameters (variant)
FIX_AXIS_PAR_LIST		Fixed axis values
FORMAT		Display format of axis points
MAX_GRAD		Maximum gradient with respect to this axis
MONOTONY		Monotony with respect to this axis
READ_ONLY		'Read Only' attribut
BIT_MASK		Bit mask to decode single-bit values
BYTE_ORDER		Byte order
COMPARISON_QUANTITY		Comparison quantity
DISPLAY_IDENTIFIER		Optional display name
EXTENDED_LIMITS		Extended limits, e.g. hard limits
FORMAT		Display format of values
FUNCTION_LIST		Function orientation
IF_DATA	(*)	Interface-specific description data
NUMBER		Number of ASCII characters or fixed values
READ_ONLY		'Read Only' attribut
COMPU_METHOD	(*)	Conversion method
COEFFS		Coefficients for fractional rational function
COMPU_TAB_REF		Reference to conversion table
FORMULA		Conversion formula
FORMULA_INV		Invers conversion formula
COMPU_TAB	(*)	Conversion table
COMPU_VTAB	(*)	Verbal conversion table
DEFAULT_VALUE		Default output string
COMPU_VTAB_RANGE	(*)	Description of range based verbal conversion tables
FRAME		Frame
FRAME_MEASUREMENT		Frame measurement objects
IF_DATA	(*)	Interface-specific description data

Keyword	(*)=multiple	Meaning
FUNCTION	(*)	Function description
ANNOTATION	(*)	Set of notes
DEF_CHARACTERISTIC		Defined adjustable objects
IN_MEASUREMENT		Input quantity
LOC_MEASUREMENT		Local quantity
OUT_MEASUREMENT		Output quantity
REF_CHARACTERISTIC		Referencedd adjustable objects
SUB_FUNCTION		Subfunction of respectiv function
GROUP	(*)	Declaration of groups
ANNOTATION	(*)	Set of notes
ROOT		Flag for root node
REF_CHARACTERISTIC		Reference to characteristic objects
REF_MEASUREMENT		Reference to measurement objects
FUNCTION_LIST		Function list
SUB_GROUP		Sub group
IF_DATA	(*)	Interface-specific description data
SOURCE	(*)	Acqusition mode
MEASUREMENT	(*)	Measurement object
ANNOTATION	(*)	Set of notes
ARRAY_SIZE		Array size of measurement objects
BIT_MASK		Bit mask to decode single-bit values
BIT_OPERATION		Bit operation
LEFT_SHIFT		Number of bit positions to shift left
RIGHT_SHIFT		Number of bit positions to shift right
SIGN_EXTEND		sign extension for measurement data
BYTE_ORDER		Byte order of measurement object
DISPLAY_IDENTIFIER		Optional display name
FORMAT		Display format of measurement object
FUNCTION_LIST		Function orientation
IF_DATA	(*)	Interface-specific description data
MULTIPLEX (ASAP1B_CAN)		Standard-ASAP1B_CAN-description: Multiplex mode
MAX_REFRESH		Refresh rate in the control unit
READ_WRITE		'Writeable'
VIRTUAL		Virtual measurement
MOD_COMMON		Module-wide (ECU specific) valid definitions
BYTE_ORDER		Byte order
DATA_SIZE		Data size in bits
DEPOSIT		Standard deposit mode for axis
S_REC_LAYOUT		Reference to the standard record layout
MOD_PAR		Control unit management data
ADDR_EPK		Address of EPROM identifier
ADDRESS_MAPPING		Address mapping
CPU_TYPE		CPU
CUSTOMER		Firm or customer
CUSTOMER_NO		Customer number
ECU		Control unit
ECU_CALIBRATION_OFFSET		Address offset
EPK		EPROM identifier
MEMORY_LAYOUT	(*)	Memory layout
IF_DATA	(*)	Interface-specific description data
MEMORY_SEGMENT	(*)	Memory segment
IF_DATA	(*)	Interface-specific description data
NO_OF_INTERFACES		Number of interfaces
PHONE_NO		Phone number of application engineer responsible
SUPPLIER		Manufacturer or supplier
SYSTEM_CONSTANT	(*)	System-defined constants
USER		User
VERSION		Module-specific version identifier

Keyword	(*)=multiple	Meaning
RECORD_LAYOUT	(*)	Description of the record layout
AXIS_PTS_X		X axis points
AXIS_PTS_Y		Y axis points
AXIS_PTS_Z		Z axis points
DIST_OP_X		X axis: parameter 'distance' for fixed characteristics
DIST_OP_Y		Y axis: parameter 'distance' for fixed characteristics
DIST_OP_Z		Z axis: parameter 'distance' for fixed characteristics
FIX_NO_AXIS_PTS_X		Fixed number of X axis points
FIX_NO_AXIS_PTS_Y		Fixed number of Y axis points
FIX_NO_AXIS_PTS_Z		Fixed number of Z axis points
FNC_VALUES		Table values
IDENTIFICATION		Identification
NO_AXIS_PTS_X		Number of X axis points
NO_AXIS_PTS_Y		Number of Y axis points
NO_AXIS_PTS_Z		Number of Z axis points
OFFSET_X		X axis: parameter 'offset' for fixed characteristics
OFFSET_Y		Y axis: parameter 'offset' for fixed characteristics
OFFSET_Z		Z axis: parameter 'offset' for fixed characteristics
RESERVED	(*)	Parameter is skipped (not interpreted)
RIP_ADDR_W		Table value: Address 'result of interpolation'
RIP_ADDR_X		X axis: Address 'result of interpolation'
RIP_ADDR_Y		Y axis: Address 'result of interpolation'
RIP_ADDR_Z		Z axis: Address 'result of interpolation'
SHIFT_OP_X		X axis: parameter 'shift' for fixed characteristics
SHIFT_OP_Y		Y axis: parameter 'shift' for fixed characteristics
SHIFT_OP_Z		Z axis: parameter 'shift' for fixed characteristics
SRC_ADDR_X		X axis: Address of input quantity
SRC_ADDR_Y		Y axis: Address of input quantity
SRC_ADDR_Z		Z axis: Address of input quantity
USER_RIGHTS	(*)	Groups with constitute access rights
REF_GROUP	(*)	List of referenced groups
READ_ONLY		Read only
VARIANT_CODING		Variant coding
VAR_CHARACTERISTIC	(*)	Definition of variant coded adjustable objects
VAR_MEASUREMENT		Measurement object which indicates criterion value
VAR_CRITERION	(*)	Definition of variant criterion
VAR_ADDRESS		Adjustable objects address list (start address of variants)
VAR_FORBIDDEN_COMB		Forbidden combinations of different variants
VAR_NAMING		Naming of variant coded adjustable objects
VAR_SEPARATOR		Separator of ajustable objects names

6.2 Predefined data types

ident

typedef char [MAX_IDENT + 1] ident;
String with *MAX_IDENT (at present = 255)* alphanumerical characters including points and brackets, interpreted as hierarchical concatenation of partial strings seperated by points. Every partial string may not exceed *MAX_PARTIAL_IDENT (at present = 32)* characters, including the length of an optional array index (numeric or as a symbolic string) in brackets at the end of the partial string. One string without a point in between is also possible, in this case MAX_IDENT = MAX_PARTIAL_IDENT. The number of partial strings within ident is not limited. The character chain must correspond with the identifier laws defined in programming language C. Identifiers can represent instances of array elements or instances of elements of complex C types or nested combinations of these. An instance of the element of a struct type would be represented by the concatenation of the instance name, a point and the element name. An instance of an array element would be represented by an instance name followed by a pair of brackets which contain either a numeric value or a symbolic string which is defined as an enumerator of an ENUM definition of the C program. Identifiers are random names which may contain characters A through Z, a through z, underscore (_), numerals 0 through 9, points (‘.’) and brackets (‘[‘,’]’) . However, the following limitations apply: the first character must be a letter or an underscore, brackets must occur in pairs at the end of a partial string and must contain a number or an alphabetic string (description of the index of an array element).

Note:
Identifiers consisting of partial identifiers seperated by points (concatenation of instance name and element name) may be presented by the MCD system in a hierarchical manner (show instance name first, then allow access to an element of the instance). This allows existing MCD systems to restrict the display length of the identifier to MAX_PARTIAL_IDENT.

Important Note:
Identifiers generally must not match to the following defined ASAP2 keywords. The ASAP2 keywords are completely listed in Appendix F.

string

typedef char [MAX_STRING + 1] string:
ANSI C compliant ‘C type’ string with maximum MAX_STRING (at present = 255) characters**.** Begin and end of the string are indicated by a double inverted comma. Only the following sequence of escape characters is allowed : "\r\n", which describes a carriage return. MCD systems may ignore the carriage return sequence and/or apply wrapping or scrolling of strings when displayed. If the string contains one or more double inverted commas, a backslash (escape char. indicator) concatenated with a double inverted comma must be inserted (example: "hello \"world\" how are you ?"). Alternatively, two double inverted commas can be inserted in this case (example: "hello ""world"" how are you ?") for compatibility with ASAP2 V1.2 and prior.

float 8-byte floating point number (IEEE format)

int 2-byte signed integer

long 4-byte signed long

datatype typedef enum datatype { UBYTE, SBYTE, UWORD, SWORD, ULONG,
SLONG, FLOAT32_IEEE, FLOAT64_IEEE }
Enumeration for description of the basic data types in the ECU program (format of FLOAT32_IEEE: see Appendix B: IEEE-Floating-Point-Format).

datasize typedef enum datasize {BYTE, WORD, LONG}
Enumeration for description of the word lengths in the ECU program

addrtype Enumeration for description of the addressing of table values or axis point values:

PBYTE: The relevant memory location has a 1 byte pointer to this table value or axis point value.

PWORD: The relevant memory location has a 2 byte pointer to this table value or axis point value.

PLONG: The relevant memory location has a 4 byte pointer to this table value or axis point value.

DIRECT: The relevant memory location has the first table value or axis point value, all others follow with incrementing address.

byteorder typedef enum byteorder { LITTLE_ENDIAN, BIG_ENDIAN,
MSB_LAST, MSB_FIRST}

Enumeration for description of the byte order in the control unit program.

Note: Use of LITTLE_ENDIAN and BIG_ENDIAN defined with keyword BYTE_ORDER leads to mistakes because it is in contradiction to general use of terms „little endian“ and „big endian“. The keywords LITTLE_ENDIAN and BIG_ENDIAN should no longer be used, they should be replaced by MSB_LAST and MSB_FIRST which are equivalent (definition of MSB_LAST and MSB_FIRST: see keyword BYTE_ORDER).

indexorder Enumeration for description of the axis point sequence in the memory.
INDEX_INCR: Increasing index with increasing address
INDEX_DECR: decreasing index with increasing address

6.3 Alphabetical list of keywords

Some individual elements of the database are delimited by ‘/begin’ and ‘/end’ keywords, similarly to the opening '{' and closing '}' brackets in 'C'. The delimiters are applied to those elements that contain an optional part, to prevent ambiguous expressions. The delimiters following defined with the ASAP2 keywords are mendatory, i.e. the delimiters have to be used if defined and mustn’t be used if not defined.

To reduce the size of description files the short delimiters '{' and '}' can be used:

<keyword> { <description_body> }

instead of:

/begin <keyword> <description_body> /end <keyword>

The short delimiters should preferably be used. The delimiters should not be mixed, i.e. don't use short delimiters and '/begin' or '/end' respectively in the same ASAP2 description file.

Note: Since version 1.31 the usage of `{` and `}` is no more recomended. None of the currently existing systems uses brackes, so the useage of brackets will probably cause problems.

6.3.1 A2ML

Prototype:

/begin A2ML	FormatSpecification
/end A2ML	

Parameters:

FormatSpecification	A2ML code for description of interface specific description data.

Description:

This keyword identifies the format description of the interface specific description data.

Example:

See page 214: file SUPP1_IF.AML and file SUPP2_IF.AML

6.3.2 ADDR_EPK

Prototype:

ADDR_EPK	Address

Parameters:

long Address:	Address of the EPROM identifier

Description:

Address of the EPROM identifier

Example:

```
ADDR_EPK              0x145678
```

6.3.3 ADDRESS_MAPPING

Prototype:

ADDRESS_MAPPING Orig_Address Mapping_Address Length

Parameters:

long Orig_Address:	Base address of the memory segment to be mapped.
long Mapping_Address:	Address to which the base address is to be mapped
long Length	Length of the segment to be mapped

Description:

Remapping of the address space of the ECU to an access address. This is needed for special application methods (for example, special address calculation is needed in KWP2000, where only 24 bit addresses are available.) The address mapping may only be used inside an IF_DATA section of a MEMORY_SEGMENT.

Example:

```
/begin IF_DATA ASAP1B_ETK
        /* ADDRESS_MAPPING    orig_addr    mapping_addr length */
          ADDRESS_MAPPING     0x4000       0x8000              0x0200
/end IF_DATA

/begin IF_DATA ASAP1B_KWP2000
        /* ADDRESS_MAPPING    orig_addr    mapping_addr length */
          ADDRESS_MAPPING     0x4000       0x6000              0x0200
/end IF_DATA
```

6.3.4 ALIGNMENT_BYTE

Prototype:

ALIGNMENT_BYTE AlignmentBorder

Parameters:

int AlignmentBorder: describes the border at which the value is aligned to, i.e. its memory address must be dividable by the value AlignmentBorder.

Description:

In complex objects (maps and axis) the alignment of a value may not coincide with the bitwidth of a value. This keyword is used to define the alignment in the case of bytes. Used in MOD_COMMON and RECORD_LAYOUT.

Example:

```
ALIGNMENT_BYTE          4 /* bytes have a 4-byte alignment */
```

6.3.5 ALIGNMENT_FLOAT32_IEEE

Prototype:

ALIGNMENT_FLOAT32_IEEE AlignmentBorder

Parameters:

int AlignmentBorder:	describes the border at which the value is aligned to, i.e. its memory address must be dividable by the value AlignmentBorder.

Description:

In complex objects (maps and axis) the alignment of a value may not coincide with the bitwidth of a value. This keyword is used to define the alignment in the case of 32bit floats.
Used in MOD_COMMON and RECORD_LAYOUT.

Example:

```
ALIGNMENT_ FLOAT32_IEEE  4 /* 32bit floats have a 4-byte alignment */
```

6.3.6 ALIGNMENT_FLOAT64_IEEE

Prototype:

ALIGNMENT_FLOAT64_IEEE AlignmentBorder

Parameters:

int AlignmentBorder: describes the border at which the value is aligned to, i.e. its memory address must be dividable by the value Alignment-Border.

Description:

In complex objects (maps and axis) the alignment of a value may not coincide with the bitwidth of a value. This keyword is used to define the alignment in the case of 64bit floats.
Used in MOD_COMMON and RECORD_LAYOUT.

Example:

```
ALIGNMENT_ FLOAT64_IEEE  4 /* 64bit floats have a 4-byte alignment */
```

6.3.7 ALIGNMENT_LONG

Prototype:

ALIGNMENT_LONG AlignmentBorder

Parameters:

int AlignmentBorder: describes the border at which the value is aligned to, i.e. its memory address must be dividable by the value Alignment-Border.

Description:

In complex objects (maps and axis) the alignment of a value may not coincide with the bitwidth of a value. This keyword is used to define the alignment in the case of longs. Used in MOD_COMMON and RECORD_LAYOUT.

Example:

```
ALIGNMENT_LONG   8 /* longs have a 8-byte alignment */
```

6.3.8 ALIGNMENT_WORD

Prototype:

ALIGNMENT_WORD AlignmentBorder

Parameters:

int AlignmentBorder: describes the border at which the value is aligned to, i.e. its memory address must be dividable by the value AlignmentBorder.

Description:

In complex objects (maps and axis) the alignment of a value may not coincide with the bitwidth of a value. This keyword is used to define the alignment in the case of words. The alignment is 2 if the parameter is missing.

Used in MOD_COMMON and RECORD_LAYOUT.

Example:

```
ALIGNMENT_WORD          4 /* words have a 4-byte alignment */
```

6.3.9 ANNOTATION

Prototype:

```
/begin ANNOTATION
                        [-> ANNOTATION_LABEL]
                        [-> ANNOTATION_ORIGIN]
                        [-> ANNOTATION_TEXT]
/end ANNOTATION
```

Parameters:

none

Optional Parameters:

-> ANNOTATION_LABEL:	label or title of the annotation
-> ANNOTATION_ORIGIN:	creator or creating system of the annotation
-> ANNOTATION_TEXT:	text of the annotation, voluminous description text

Description:

One ANNOTATION may represent a voluminous description. Its purpose is to be e.g. an application note which explains the function of an identifer for the calibration engineer.
Note : An ANNOTATION may occur several times within a definition (due to compatibility with MSR/MEDOC SW-DTD, the future ASAP2 V2.0).

Example:

```
/begin CHARACTERISTIC  annotation.example1
   ....
 /begin ANNOTATION
   ANNOTATION_LABEL "Luftsprungabhängigkeit"
   ANNOTATION_ORIGIN "Graf Zeppelin"
   /begin ANNOTATION_TEXT
     "Die luftklasseabhängigen Zeitkonstanten t_hinz\r\n“
     "& t_kunz können mit Hilfe von Luftsprüngen ermittelt werden.\r\n“
     "Die Taupunktendezeiten in großen Flughöhen sind stark “
     "schwankend.“
   /end ANNOTATION_TEXT
 /end ANNOTATION

 /begin ANNOTATION
   ANNOTATION_LABEL "Taupunktendezeiten"
```

```
     /begin ANNOTATION_TEXT
          "Flughöhe    Taupunktendezeit\r\n"
           " 13000ft      20 sec\r\n"
           " 25000ft      40 sec\r\n"
           " 35000ft      12 sec"
      /end ANNOTATION_TEXT
  /end ANNOTATION

....
/end CHARACTERISTIC
```

6.3.10 ANNOTATION_LABEL

Prototype:

ANNOTATION_LABEL label

Parameters:

string	label	label or title of the annotation

Description:

Assign a title to an annotation. Useful as a definition can contain more than one annotation. Recommendation : The ANNOTATION_LABEL shall describe the use-case of the ANNOTATION, e.g. „Calibration Note“.

Example:

ANNOTATION_LABEL "Calibration Note“

6.3.11 ANNOTATION_ORIGIN

Prototype:

ANNOTATION_ORIGIN origin

Parameters:

string	origin	creator or creating system of the annotation

Description:

To identify who or which system has created an annotation.

Example:

```
ANNOTATION_ORIGIN "from the calibration planning department“
```

6.3.12 ANNOTATION_TEXT

Prototype:

```
/begin ANNOTATION_TEXT   {annotation_text}*
/end ANNOTATION_TEXT
```

Parameters:

string	annotation_text

Description:

One ANNOTATION_TEXT may represent a multi-line ASCII description text (voluminous description). Its purpose is to be an application note which explains the function of an identifer for the calibration engineer.

Example:

```
/begin CHARACTERISTIC  annotation.example2 ...
   ....
     /begin ANNOTATION
          ANNOTATION_LABEL  "Calibration Note“
          /begin ANNOTATION_TEXT
               "Blariblara plumpaquatsch. Oder sonst ein beliebiger “
                "Text.\r\n“
      "Lediglich, wenn ein Doppelhochkomma “
                "auftritt, muß man das als \“ oder ““ markieren.“
         /end ANNOTATION_TEXT
     /end ANNOTATION
....
/end CHARACTERISTIC
```

6.3.13 ARRAY_SIZE

Prototype:

ARRAY_SIZE Number

Parameters:

int Number: Number of measurement values included in respective measurement object (maximum value of ‘Number’: 32767).

Description:

This keyword marks a measurement object as an array of <Number> measurement values.

Example:

```
/begin MEASUREMENT         N               /* name */
                           "Engine speed"  /* long identifier */
                           UWORD           /* datatype */
                           R_SPEED_3       /* conversion */
                           2               /* resolution */
                           2.5             /* accuracy */
                           120.0           /* lower limit */
                           8400.0          /* upper limit */
        ARRAY_SIZE         8               /* array of 8 values */
        BIT_MASK           0x0FFF
        BYTE_ORDER         MSB_FIRST
        /begin FUNCTION_LIST ID_ADJUSTM  FL_ADJUSTM
        /end FUNCTION_LIST
        /begin IF_DATA     ISO SND 0x10 0x00 0x05 0x08 RCV 4 long
        /end IF_DATA
/end MEASUREMENT
```

6.3.14 A2ML_VERSION

Prototype:

A2ML_VERSION Version Upgrad

Parameters:

int VersionNo:	Version number of AML part
int UpgradNo:	Upgrade number of AML part

Description:

The keyword A2ML_VERSION is new in ASAP2 version 1.31. The reason for this keyword is, to declare what kind of BLOBs should be generated from the A2ML parts. Since ASAP2 version 1.31 we have a specification for the storage layout of the BLOBs. The keyword is optional. When the keyword is omitted, or the version number is below 1.31 then the ASAP2 parser may produce the old BLOB format. When the A2ML version number is 1.31, then the new format must be generated.

The A2ML version can be expressed by two numerals:
- VersionNo
- UpgradNo

where 'VersionNo' represents the main version number and 'UpgradNo' the upgrade number (fractional part of version number). The upgrade number will be incremented if additional functionality is added to ASAP2 Meta Language standard which has no effect on existing applications (compatible modifications). The version number will be incremented in case if incompatible modifications.

Example:

```
A2ML_VERSION        1  31               /* Version 1.31 */
```

6.3.15 ASAP2_VERSION

Prototype:

ASAP2_VERSION Version Upgrad

Parameters:

int VersionNo:	Version number of ASAP2 standard
int UpgradNo:	Upgrade number of ASAP2 standard

Description:

The ASAP2 version can be expressed by two numerals:

- VersionNo
- UpgradNo

where ‘VersionNo’ represents the main version number and ‘UpgradNo’ the upgrade number (fractional part of version number). The upgrade number will be incremented if additional functionality is implemented to ASAP2 standard which has no effect on existing applications (compatible modifications). The version number will be incremented in case if incompatible modifications.

Example:

```
ASAP2_VERSION      1  31               /* Version 1.31 */
```

6.3.16 AXIS_DESCR

Prototyp:

```
/begin AXIS_DESCR         Attribute InputQuantity Conversion MaxAxisPoints
                          LowerLimit UpperLimit
                          [-> READ_ONLY]
                          [-> FORMAT]
                          {-> ANNOTATION} *
                          [-> AXIS_PTS_REF]
                          [-> MAX_GRAD]
                          [-> MONOTONY]
                          [-> BYTE_ORDER]
                          [-> EXTENDED_LIMITS]
                          [-> FIX_AXIS_PAR]
                          [-> FIX_AXIS_PAR_DIST]
                          [-> FIX_AXIS_PAR_LIST]
                          [-> DEPOSIT]
/end AXIS_DESCR
```

Parameters:

enum Attribute: Description of the axis points:

STD_AXIS: Standard axis

FIX_AXIS: This is a curve or a map with virtual axis points that are not deposited at EPROM. The axis points can be calculated from parameters defined with keywords FIX_AXIS_PAR, FIX_AXIS_PAR_DIST and FIX_AXIS_PAR_LIST. The axis points can‘t be modified.

COM_AXIS: Group axis points or description of the axis points for SIEMENS deposit. For this variant of the axis points the axis point values are separated from the table values of the curve or map in the emulation memory and must be described by a special AXIS_PTS data record. The reference to this record occurs with the keyword 'AXIS_PTS_REF'.

RES_AXIS: Rescale axis. For this variant of the axis points the axis point values are separated from the table values of the curve or map in the emulation memory and must be described by a special AXIS_PTS data record. The reference to this record occurs with the keyword 'AXIS_PTS_REF'.

ident InputQuantity: Reference to the data record for description of the input quantity (see MEASUREMENT). If there is no input quantity assigned, parameter 'InputQuantity' should be set to "NO_INPUT_QUANTITY" (application systems must be capable to treat this case).

ident Conversion: Reference to the relevant record of the description of the conversion method (see COMPU_METHOD).

int MaxAxisPoints : Maximum number of axis points
Note: The application system can change the dimensions of a characteristic (increase or decrease the number of axis points). The number of axis points may not be increased at random as the address range reserved for each characteristic in the ECU program by the application system cannot be changed.

float LowerLimit: Plausible range of axis point values, lower limit

float UpperLimit: Plausible range of axis point values, upper limit

Optional parameters:

-> READ_ONLY: This keyword can be used to indicate that the axis points of adjustable object cannot be changed (but can be read only).
Note: This optional keyword used at CHARACTERISTIC record indicates the adjustable object to be read only at all (table values and axis points).

-> FORMAT: With deviation from the display format specified with keyword COMPU_TAB referenced by parameter <Conversion> a special display format can be specified to be used to display the axis points.

-> AXIS_PTS_REF: Reference to the AXIS_PTS record for description of the axis points distribution.

-> MAX_GRAD: This keyword can be used to specify a maximum permissible gradient for the adjustable object with respect to this axis (MaxGrad= max (abs(($W_{i,k}$-$W_{i-1,k}$)/(X_i-X_{i-1})))).

-> MONOTONY: This keyword can be used to specify a monotonous behaviour for the adjustable object with respect to this axis.

-> BYTE_ORDER: Where the standard value does not apply this parameter can be used to specify the byte order (Intel format, Motorola format) of the axis point value.

-> FIX_AXIS_PAR: For curves or maps, the axis points distribution is not stored in memory but it is computed on the basis of the offset (initial value) and a difference. For the record layouts used today, these parameters must be included in the description file. The specification occurs with keyword 'FIX_AXIS_PAR'.

-> FIX_AXIS_PAR_DIST: Similar to FIX_AXIS_PAR but with a different computing method.

-> FIX_AXIS_PAR_LIST: The original values of the axis are directly contained in the file. The assigned COMPU_METHOD is applied to achieve the actual display values from the values with this keyword.

-> DEPOSIT: The axis points of a characteristic can be deposited in two different ways:

a) The individual axis point values are deposited as absolute values.

b) The individual axis point are stored as differences. Each axis point value is determined from the adjacent axis point (predecessor).

Where the standard value does not appply this parameter can be used to specify the axis point deposit.

-> ANNOTATION: Set of notes (represented as multi-line ASCII description texts) which are related. Can serve e.g. as application note. When a COM_AXIS is referenced it is sufficient to place the ANNOTATION with its AXIS_PTS in order to avoid redundant information.

-> EXTENDED_LIMITS: This keyword can be used to specify an extended range of values. In the application system, for example, when leaving the standard range of values (lower limit...upper limit) a warning could be generated (extended limits enabled only for "power user").

Description:

Axis description within an adjustable object

Notes:

For the BOSCH group characteristics and for the SIEMENS standard curves or maps, the mandatory parameters 'input quantity', 'conversion','max axis points', 'lower limit', and 'upper limit' can be specified with the keyword AXIS_DESCR as well as with the keyword AXIS_PTS. This redundancy has been introduced to simplify the processing program. For these redundant parameters the processing programs should include a consistency check.

With the 'input quantity' parameter reference is made to a measurement object (MEASUREMENT, Page 131). The MEASUREMENT keyword also specifies the 'conversion', 'lower limit' and 'upper limit' parameters.

These parameters are not always identical to the parameters specified under AXIS_DESCR, as for the relevant measurement object either another conversion method (e.g. other solution) or other plausibility limits can apply than for the axis points of the relevant characteristic.

Note: The keywords FIX_AXIS_PAR, FIX_AXIS_PAR_DIST, DEPOSIT and FIX_AXIS_PAR_LIST are mutually exclusive, i.e. please use at most one of these keywords at the same AXIS_DESCR record.

Example:

```
/begin AXIS_DESCR        STD_AXIS        /* Standard axis points */
                         N               /* Reference to input quantity */
                         CONV_N          /* Conversion */
                         14              /* Max.number of axis points*/
                         0.0 5800.0      /* Upper limit, lower limit*/
        MAX_GRAD         20.0            /* Axis: maximum gradient*/
/end AXIS_DESCR
```

6.3.17 AXIS_PTS

Prototype:

```
/begin AXIS_PTS           Name LongIdentifier Address InputQuantity Deposit MaxDiff
                          Conversion MaxAxisPoints LowerLimit UpperLimit
                          [-> DISPLAY_IDENTIFIER]
                          [-> READ_ONLY]
                          [-> FORMAT]
                          [-> DEPOSIT]
                          [-> BYTE_ORDER]
                          [-> FUNCTION_LIST]
                          [-> REF_MEMORY_SEGMENT]
                          [-> GUARD_RAILS]
                          [-> EXTENDED_LIMITS]
                          {-> ANNOTATION} *
                          {-> IF_DATA } *
/end AXIS_PTS
```

Parameters:

ident Name: unique identifier in the ECU program ***(must be unique within the ASAP2 MODULE)***.

string LongIdentifier: comment, description

long Address: address of the adjustable object in the emulation memory

ident InputQuantity: reference to the data record for description of the input quantity (see MEASUREMENT). If there is no input quantity assigned, parameter 'InputQuantity' should be set to "NO_INPUT_QUANTITY" (application systems must be capable to treat this case).

ident Deposit: reference to the relevant data record for description of the record layout (see RECORD_LAYOUT)

float MaxDiff: maximum float with respect to the adjustment of a table value

ident Conversion: reference to the relevant data record for description of the conversion method (see COMPU_METHOD)

int MaxAxisPoints: maximum number of axis points

float LowerLimit: plausible range of axis point values, lower limit

float UpperLimit: plausible range of axis point values, upper limit

Optional parameters:

-> DISPLAY_IDENTIFIER**:** Can be used as a display name (alternative to the ‘name’ attribute).

-> READ_ONLY: This keyword can be used to indicate that the axis points of axis points distribution cannot be changed (but can be read only).

Note: This optional keyword used at CHARACTERISTIC record indicates the adjustable object to be read only at all (table values and axis pts).

-> FORMAT: With deviation from the display format specified with keyword COMPU_TAB referenced by parameter <Conversion> a special display format can be specified to be used to display the axis points.

-> DEPOSIT: The axis points of a characteristic can be deposited in one of the following two modes:

a) the individual axis points are deposited as absolute values;
b) the individual axis points are deposited as differences. Each axis point is determined from the adjacent point (predecessor). Where the standard value does not apply, this parameter can be used to specify the deposit of axis points.

-> BYTE_ORDER: Where the standard value does not apply, this parameter can be used to specify the byte order (Intel format, Motorola format) of the axis points.

-> FUNCTION_LIST: This keyword can be used to specify a list of 'functions' to which the axis points distribution is allocated (function orientation).

Remark: Since ASAP2 version 1.20 the keyword FUNCTION comprises some additional features to describe functional structure and dependencies. The keyword FUNCTION_LIST is going to be canceled at ASAP2 version 2.00.

-> REF_MEMORY_SEGMENT: Reference to the memory segment which is needed if the address is not unique (this occurs in the case of lapping address ranges (overlapping memory segments).

-> GUARD_RAILS: This keyword is used to indicate that an AXIS_PTS uses guard rails. The Measurement and Calibration System does not allow the user to edit the outermost axis breakpoints (see GUARD_RAILS).

-> IF_DATA: Data record for description of the interface specific description data (BLOB: binary large object). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the control unit supplier or interface module supplier. The structure of this data record corresponds with the structure of the interface-specific description data of the CHARACTERISTIC data record.

-> ANNOTATION: Set of notes (represented as multi-line ASCII description texts) which are related. Can serve e.g. as application note.

-> EXTENDED_LIMITS: This keyword can be used to specify an extended range of values. In the application system, for example, when leaving the standard range of values (lower limit...upper limit) a warning could be generated (extended limits enabled only for "power user").

Description:

AXIS_PTS

Specification of parameters for the handling of an axis points distribution.

Example:

```
/begin AXIS_PTS             STV_N           /* name */
                            "axis points disrtribution speed"      /* long identifier */
                            0x9876          /* address */
                            N               /* input quantity */
                            DAMOS_SST       /* deposit */
                            100.0           /* maxdiff */
                            R_SPEED         /* conversion */
                            21              /* maximum number of axis points */
                            0.0  5800.0     /* lower and upper limit */
                            GUARD_RAILS /* uses guard rails*/
      REF_MEMORY_SEGMENT Data3
      /begin FUNCTION_LIST  ID_ADJUSTM  FL_ADJUSTM SPEED_LIM
      /end FUNCTION_LIST
      /begin IF_DATA        DIM EXTERNAL DIRECT
      /end IF_DATA
/end AXIS_PTS
```

6.3.18 AXIS_PTS_REF

Prototype:

AXIS_PTS_REF AxisPoints

Parameters:

ident AxisPoints: Name of the AXIS_PTS data record which describes the axis points distribution (group axis points and SIEMENS record layout: see AXIS_PTS).

Description:

In the BOSCH adjustable types 'group characteristic curve' and 'group characteristic map' and in the SIEMENS record layout, the addresses of the axis point values are separated from the table values in the emulation memory and must be described by a special AXIS_PTS data record. This data record is referenced by means of the keyword AXIS_PTS_REF.

Example:

```
/* Group characteristic curve with reference to axis points distribution GRP_N */
/begin CHARACTERISTIC     TORQUE              /* name */
                          "Torque limitation" /* long identifier */
                          CURVE 0x1432        /* type, address */
                          DAMOS_GKL 0.2       /* deposit, maxdiff */
                          R_TORQUE            /* conversion */
                          0.0  43.0           /* lower limit, upper limit */
      /begin IF_DATA      DIM EXTERNAL INDIRECT /end IF_DATA
      /begin AXIS_DESCR   /* description of X-axis points */
                          COM_AXIS    N       /* common axis points, input quantity */
                          CONV_N 14           /* conversion, max. no. of axis p.*/
                          0.0  5800.0         /* lower limit, upper limit */
         AXIS_PTS_REF     GRP_N
      /end AXIS_DESCR
/end CHARACTERISTIC

/* Axis points distribution data record */
/begin AXIS_PTS           GRP_N               /* name */
                          "Group axis points speed" / * long identifier */
                          0x1032  N           /* address, input quantity */
                          DAMOS_GST           /* deposit */
                          50.0   CONV_N       /* maxdiff, conversion */
                          11                  /* max. no. of axis points */
                          0.0   5800.0        /* lower limit, upper limit */
      /begin IF_DATA      DIM EXTERNAL INDIRECT /end IF_DATA
/end AXIS_PTS
```

6.3.19 AXIS_PTS_X/_Y/_Z

Prototype:

AXIS_PTS_X/ _Y/ _Z Position Datatype IndexIncr Addressing

Parameters:

int Position:	Position of the axis point values in the deposit structure (description of sequence of elements in the data record).
datatype Datatype:	Data type of the axis point values
indexorder IndexIncr:	Decreasing or increasing index with increasing addresses
addrtype Addressing:	Addressing of the table values (see enum addrtype).

Description:

Description of the X axis points or Y axis points in the memory (see keyword RECORD_LAYOUT)

Example:

```
AXIS_PTS_X                3 ULONG INDEX_INCR DIRECT
```

6.3.20 AXIS_RESCALE_X/_Y/_Z

Prototype:

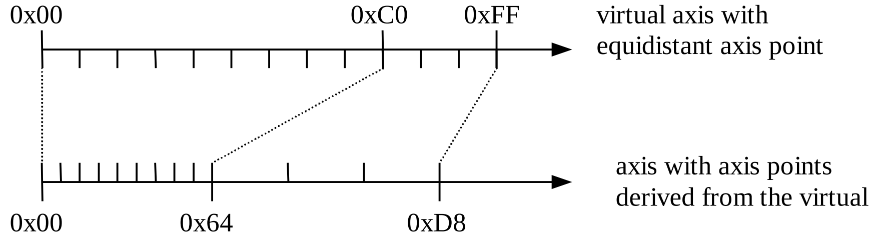
AXIS_RESCALE_X/_Y/_Z Position Datatype MaxNumberOfRescalePairs IndexIncr Adressing

Parameters:

int Position: position of the rescale axis point value pairs in the deposit structure (description of sequence of elements in the data record).
datatype DataType: Data type of the rescale axis point values
int MaxNumberOfRescalePairs:
maximum number of rescaling axis point pairs
(see NO_RESCALE_PTS_X/_Y/_Z)
indexorder IndexIncr: Decreasing or increasing index with increasing addresses
addrtype Adressing: Addressing of the table values (see enum addrtype).

Description:

Description of rescaling the axis values of an adjustable object. A rescale axis consists mainly of a number of rescaling axis points pairs (*axis*$_i$, *virtual*$_i$) which describe a rescale mapping between the axis points and a virtual axis that is used for the access of the table function values deposited in the control unit. Between two pairs the mapping is linear. Both, the axis points and the virtual axis points must be in ascending order. Consider, for example, the three rescale pairs (0x00, 0x00), (0x64, 0xC0) and (0xD8, 0xFF). Then all axis points between 0x00 and 0x64 are mapped linear to the virtual axis [0x00, 0xC0], and all axis points between 0x64 and 0xD8 are mapped linear to the virtual axis [0xC0, 0xFF]:



Accordingly, to each axis point there is a virtual axis point. The virtual axis points are distributed equidistantly on the virtual axis including the axis limits, e.g. the virtual axis points can be derived from the size of the virtual axis and the number of axis points. According to the rescale mapping the axis point can be computed from the virtual axis points. The following algorithm can be applied, where D is the length of the (equidistant) intervals on virtual axis:

$$D = \frac{\text{last virtual axis point - first virtual axis point + 1}}{no\_axis\_pts - 1} \qquad k = 1$$

```
FOR i = 1 TO $\left(no\_rescale\_x - 1\right)$

    FOR $k * d + virtual_1 < virtual_{i+1}$
    /* repeat for the number of points in the interval on the virtual axis */

    $k = k + 1$
    $X_k = axis_i + \left(\left(k - 1\right)D - virtual_i\right)\frac{axis_{i+1} - axis_i}{virtual_{i+1} - virtual_i}$

$X_1 = axis_1$
$X_{no\_axis\_pts} = axis_{no\_rescale\_x}$
```

It is recommended that D is a power of 2, i.e. if the size of the virtual axis is 256, the number of axis points should be $no\_axis\_pts = 2^n + 1$ = {3, 5, 9, 17, 33}.

The following example makes clear how the evaluation of the formula can be used to derive the actual axis points. We have no_of_rescale_pairs = 3 and virtual$_1$ = 0x00 = 0, virtual$_2$ = 0xC0 = 192, virtual$_3$ = 0xFF = 255, axis$_1$ = 0x00 = 0, axis$_2$= 0x64 = 100, axis$_3$ = 0xD8 = 216. Assume no_axis_pts = 9, and therefore D = 32. The first of the two executions of the inner loop (j-loop) is on virtual$_2$ – virtual$_1$/ D = 192/32 = 6 iterations. For each iteration (axis$_2$ – axis$_1$)/(virtual$_2$ – virtual$_1$) = 100/192, and therefore

X_2 = 0 + 32 * 100/192 = 16,666,
X_3 = 0 + 64 * 100/192 = 33,333,
X_4 = 0 + 96 * 100/192 = 50,
X_5 = 0 + 128 * 100/192 =66,666,
X_6 = 0 + 160 * 100/192 = 83,333.

For the second execution there are virtual$_3$ – virtual$_2$ / D = 2 iterations with (axis$_3$ – axis$_2$)/(virtual$_3$ – virtual$_2$) = 116/64. Consequently

X_7 = 100 + (192 – 192) * 116/64 = 100 and
X_8 = 100 + (224 – 192) * 116/64 = 158.

Also X_1 = axis$_1$ = 0 and X_9 = axis$_3$ = 216.

Used in RECORD_LAYOUT.

Example:

AXIS_RESCALE_X 3 UBYTE 5 INDEX_INCR DIRECT

6.3.21 BIT_MASK

Prototype:

BIT_MASK	Mask

Parameters:

long Mask:	mask to mask out single bits

Description:

The BIT_MASK keyword can be used to mask out single bits of the value to be processed.

Example:

BIT_MASK	0x00000FFF

6.3.22 BIT_OPERATION

Prototype:

```
/begin BIT_OPERATION
                                   [-> LEFT_SHIFT]
                                   [-> RIGHT_SHIFT]
                                   [-> SIGN_EXTEND]
/end BIT_ OPERATION
```

Parameters:

Optional parameters

-> LEFT_SHIFT: Number of positions to left shift data, zeros will be shifted in from the right.

-> RIGHT_SHIFT: Number of positions to right shift data, zeros will be shifted in from the left.

-> SIGN_EXTEND: Gives a sign extention of sign bit for measurement data.

Description:

The BIT_OPERATION keyword can be used to perform operation on the masked out value.
First BIT_MASK will be applied on measurment data, then LEFT_SHIFT/(RIGHT_SHIFT) is performed and last the SIGN_EXTEND is carried out.

SIGN_EXTEND means that the sign bit (masked data's leftmost bit) will be copied to all bit positions to the left of the sign bit. This results in a new datatype with the same signed value as the masked data.

Example:

```
/begin BIT_OPERATION
                                    RIGHT_SHIFT  4          /*4 positions*/
                                    SIGN_EXTEND
/end BIT_OPERATION
```

Explanation	Data	Comment
Data after mask operation	0000000000100000	
Data after shift operation	0000000000000010	shifted right 4 positions
Data after sign extend	1111111111111110	

6.3.23 BYTE_ORDER

Prototype:

BYTE_ORDER ByteOrder

Parameters:

byteorder ByteOrder: Byte order of the relevant quantity in the ECU program

Note: Use of LITTLE_ENDIAN and BIG_ENDIAN defined with keyword BYTE_ORDER in version 1.0 leads to mistakes because it is in contradiction to general use of terms „little endian“ and „big endian“. Since version 1.2 the keywords LITTLE_ENDIAN and BIG_ENDIAN are permissible but should not longer be used. They should be replaced by MSB_LAST and MSB_FIRST which are equivalent: MSB_LAST corresponds to the Intel format (equivalent former keyword is BIG_ENDIAN).

MSB_FIRST corresponds to the Motorola format (equivalent former keyword is LITTLE_ENDIAN).

Description:

Where the standard value does not apply this parameter can be used to specify the byte order (Intel format, Motorola format).

Example:

BYTE_ORDER MSB_LAST

Byte Order	Keyword	Former Keyword	Increasing address -->				
			n	n+1	....	n + (N-1)	n + N
Motorola Format	MSB_FIRST	LITTLE_ENDIAN	Byte$_N$ (Most Significant Byte)	Byte$_{N-1}$	....	Byte$_1$	Byte$_0$ (Least Significant Byte)
Intel Format	MSB_LAST	BIG_ENDIAN	Byte$_0$ (Least Significant Byte)	Byte$_1$	....	Byte$_{N-1}$	Byte$_N$ (Most Significant Byte)

Table 3: Byte order - memory data deposition

6.3.24 CALIBRATION_HANDLE

Prototype:

```
/begin CALIBRATION_HANDLE                    (Handle)*
/end CALIBRATION_HANDLE
```

Parameters:

long Handle	Handle for the calibration method

Description:

Definition of the calibration method specific. The interpretation of this data depends on the calibration method used. Used in CALIBRATION_METHOD

Example:

```
/begin CALIBRATION_HANDLE
        0x10000         /* start address of pointer table */
        0x200           /* length of pointer table */
        0x4             /* size of one pointer table entry */
        0x30000         /* start address of flash section */
        0x20000         /* length of flash section */
/end CALIBRATION_HANDLE
```

6.3.25 CALIBRATION_METHOD

Prototype:

```
/begin CALIBRATION_METHOD                Method Version
                                         [-> CALIBRATION_HANDLE]
/end CALIBRATION_METHOD
```

Parameters:

string Method: the string identifies the calibration method to be used. A convention regarding the meaning of the calibration methods. The following strings are already in use: ‘InCircuit’, ‘SERAM’, ‘DSERAP’, ‘BSERAP’

long Version Version number of the method used

Optional Parameters:

-> CALIBRATION_HANDLE Contains the (method specific) arguments for the calibration method. The arguments themselves and their meaning are dependent of the calibration method.

Description:

This keyword is used to indicate the different methods of access that are implemented in the ECU and that can be used regardless of the actual interface of the ECU.
Used in MOD_PAR.

Example:

```
/begin CALIBRATION_METHOD
        „InCircuit“
        2
        /begin CALIBRATION_HANDLE
                        0x10000         /* start address of pointer table */
                        0x200           /* length of pointer table */
                        0x4             /* size of one pointer table entry */
                        0x10000         /* start address of flash section */
                        0x10000         /* length of flash section */
        /end CALIBRATION_HANDLE
/end CALIBRATION_METHOD
```

6.3.26 CHARACTERISTIC

Prototype:

```
/begin CHARACTERISTIC      Name LongIdentifier Type Address Deposit MaxDiff
                           Conversion LowerLimit UpperLimit
                           [-> DISPLAY_IDENTIFIER]
                           [-> FORMAT]
                           [-> BYTE_ORDER]
                           [-> BIT_MASK]
                           [-> FUNCTION_LIST]
                           [-> NUMBER]
                           [-> EXTENDED_LIMITS]
                           [-> READ_ONLY]
                           [-> GUARD_RAILS]
                           [-> MAP_LIST]
                           [-> MAX_REFRESH]
                           [-> DEPENDENT_CHARACTERISTIC]
                           [-> VIRTUAL_CHARACTERISTIC]
                           [-> REF_MEMORY_SEGMENT]
                           {-> ANNOTATION} *
                           [-> COMPARISON_QUANTITY]
                           {-> IF_DATA } *
                           {-> AXIS_DESCR } *
/end CHARACTERISTIC
```

Parameters:

ident Name: ***unique identifier in the ECU program (must be unique within the ASAP2 MODULE)***

string LongIdentifier: comment, description

enum Type: possible types:
VALUE
CURVE
MAP
CUBOID
VAL_BLK (array of values)
ASCII (string)

long Address: address of the adjustable object in the emulation memory

ident Deposit: reference to the corresponding data record for description of the record laout (see RECORD_LAYOUT)

float Maxdiff: maximum float with respect to an adjustment of a table value

ident Conversion: reference to the relevant data record for description of the conversion method (see COMPU_METHOD).

float LowerLimit: plausible range of table values, lower limit

float UpperLimit: plausible range of table values, upper limit

Optional parameters

-> DISPLAY_IDENTIFIER: Can be used as a display name (alternative to the 'name' attribute).

-> FORMAT: With deviation from the display format specified with keyword COMPU_TAB referenced by parameter <Conversion> a special display format can be specified to be used to display the table values.

-> BYTE_ORDER: Where the standard value does not apply this parameter can be used to specify the byte order (Intel format, Motorola format) if the standard value is not to be used.

-> BIT_MASK: This parameter can be used to specify a bit mask for the handling of single bits.

-> FUNCTION_LIST: This keyword can be used to specify a list of 'functions' to which the relevant adjustable object is allocated (function orientation).

Remark: Since ASAP2 version 1.20 the keyword FUNCTION comprises some additional features to describe functional structure and dependencies. The keyword FUNCTION_LIST is going to be canceled at ASAP2 version 2.00.

-> NUMBER: For the adjustable object types 'fixed value block' (VAL_BLK) and 'string' (ASCII), this keyword specifies the number of fixed values and characters respectively.

-> EXTENDED_LIMITS: This keyword can be used to specify an extended range of values. In the application system, for example, when leaving the standard range of values (lower limit...upper limit) a warning could be generated (extended limits enabled only for "power user").

-> READ_ONLY: This keyword can be used to indicate that the adjustable object cannot be changed (but can be read only). This keyword indicates the adjustable object to be read only at all (table values and axis points). The optional keyword used at AXIS_DESCR record indicates the related axis points to be read only.

-> GUARD_RAILS: This keyword is used to indicate that an adjustable CURVE or MAP uses guard rails. The Measurement and Calibration System does not allow the user to edit the outermost values of the adjustable object (see GUARD_RAILS).

-> MAP_LIST: For the adjustable object type CUBOID which are `sliced', this keyword specifies the MAPs which comprise the cuboid.

-> MAX_REFRESH: Maximum refresh rate of this (adaptive) characteristic in the control unit. The existence of the keyword implies that the value of the characteristic is changed by the control unit (adaptive characteristics).

-> DEPENDENT_CHARACTERISTIC: Describes the formula and references to characteristics, upon which this characteristic depends on. Note: The dependence graph described by the dependence relation must be acyclic. This must be ensured by the producer of the ASAP2 file. This keyword is only valid for characteristics of type VALUE

-> VIRTUAL_CHARACTERISTIC: Marks a characteristic as being virtual, i.e. not existing in the memory of the control unit. The address can therefore be ignored for virtual characteristic. Initial value of the virtual characteristic depends on the values of other characteristic.
Note: The corresponding graph (in analogy to the dependence graph) must also be acyclic and each sink of the graph must be a non virtual characteristic. This must be ensured by the producer of the ASAP2 file. This keyword is only valid for characteristics of type VALUE.

-> REF_MEMORY_SEGMENT: Reference to the memory segment which is needed if the address is not unique (this occurs in the case of lapping address ranges (overlapping memory segments).

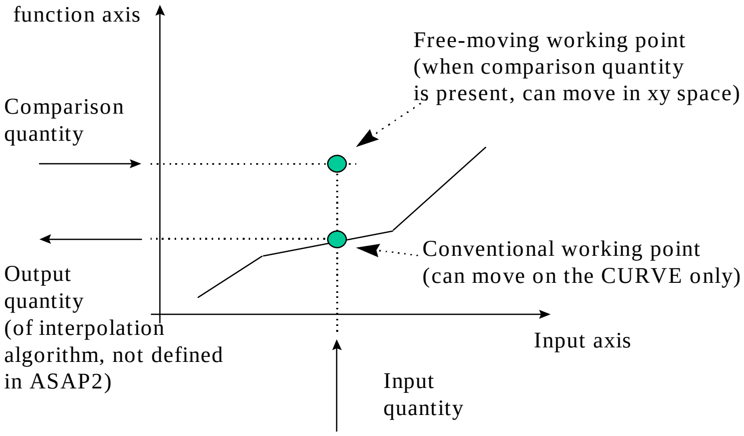
-> IF_DATA: Date record to describe the interface specific description data (BLOB:binary large object). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the control unit supplier or the interface module supplier.

-> AXIS_DESCR This keyword is used to specify the parameters for the axis description (with characteristic curves and maps). The first parameter block describes the X-axis, the second parameter block the Y-axis.

-> ANNOTATION: Set of notes (represented as multi-line ASCII description texts) which are related. Can serve e.g. as application note.

-> COMPARISON_QUANTITY: This keyword references a valid MEASUREMENT in the ASAP2 file. Semantic Interpretation (for a CURVE, a CHARACTERISTIC with only one AXIS_DESC) : The conventional workpoint for a -CURVE has only one input quantity (assigned to AXIS_DESCR) and moves on the CURVE. The 'free-moving' workpoint in an xy diagram of a CURVE is described by two quantities (the conventional input quantity with the AXIS_DESC, the x-axis, and an additional comparison quantity described as an optional attribute directly with the CURVE, the y-axis).The 'free-moving' workpoint does not move on the CURVE, but on the xy-diagram in which the CURVE is located. The crossing of the free-moving workpoint and the CURVE would describe an EVENT. Such display is required by calibration engineers of automatic transmission control (EVENT=gear shift). When this keyword with a CURVE is present, the workpoint display of the MCD system shall apply the INPUT_QUANTITY and the COMPARISON_QUANTITY in the xy-diagram.

CHARACTERISTIC


function axis
Free-moving working point
(when comparison quantity
is present, can move in xy space)
Comparison
quantity
Conventional working point
(can move on the CURVE only)
Output
quantity
(of interpolation
algorithm, not defined
in ASAP2)
Input axis
Input
quantity

Description:

Specification of the parameters for the processing of an adjustable object.

Example:

```
/begin CHARACTERISTIC       PUMKF           /* name */
                            "Pump characteristic map“          /* long identifier */
                            MAP             /* type */
                            0x7140          /* address */
                            DAMOS_KF        /* deposit */
                            100.0           /* maxdiff */
                            R_VOLTAGE       /* conversion */
                            0.0   5000.0    /* lower limit, upper limit */
    MAX_REFRESH             3 15 /* 15 msec */
    /begin DEPENDENT_CHARACTERISTIC "sin(X1)“ ALPHA
    /end DEPENDENT_CHARACTERISTIC
    /begin VIRTUAL_CHARACTERISTIC „sqrt(X1)“ B_AREA
    /end VIRTUAL_CHARACTERISTIC
    REF_MEMORY_SEGMENT Data1
    /begin FUNCTION_LIST    NL_ADJUSTMENT FL_ADJUSTMENT SPEED_LIM
    /end FUNCTION_LIST
    /begin IF_DATA          DIM EXTERNAL INDIRECT /end IF_DATA
    /begin AXIS_DESCR       /* description of X-axis points */
                            STD_AXIS        /* standard axis points */
                            N               /* reference to input quantity */
                            CON_N           /* conversion */
                            13              /* maximum number of axis points */
                            0.0   5800.0    /* lower limit, upper limit */
        MAX_GRAD            20.0            /* X-axis: maximum gradient */
    /end AXIS_DESCR
    /begin AXIS_DESCR       /* description of Y-axis points */
                            STD_AXIS        /* standard axis points */
                            AMOUNT          /* reference to input quantity */
                            CON_ME          /* conversion */
                            17              /* maximum number of axis points */
                            0.0  43.0       /* lower limit, upper limit */
    /end AXIS_DESCR
/end CHARACTERISTIC
```

6.3.27 CHECKSUM

Prototype:

```
/begin CHECKSUM            ChecksumDll
/end CHECKSUM
```

Parameters:

string ChecksumDll: Reference to DLL file name containing the ECU checksum algorithm.

Description:

Description of the checksum algorithm implemented in the ECU. The ECU supplier provides a DLL.

Note : This keyword is only used in the context of the IF_DATA(MODULE) applied for the ASAP1a-CCP.

Note : The usage of the DLL is based on a standard API definition published in the Annex.

Example:

```
/begin  CHECKSUM           "crc16.dll"
/end CHECKSUM
```

6.3.28 COEFFS

Prototype:

COEFFS	a b c d e f

Parameters:

float a, b, c, d, e, f:	coefficients for the specified formula: f(x) = (axx + bx + c) / (dxx + ex + f)

Description:

Specification of coefficients for the formula f(x) = (axx + bx + c) / (dxx + ex + f). This term describes the conversion from physical values to control unit internal values:

INT = f(PHYS);

Important: For these coefficients restrictions have to be defined because this general equation cannot always be inverted.

Example:

```
COEFFS                  0 4 8 0 0 5
/* Control unit internal values of revolutions (INT) is calculated  from     */
/* physical values (PHYS: unit of PHYS is [rpm]) as follows:                  */
/*                      INT = (4/5) * PHYS/[rpm] + (8/5)                      */
/* inverted:            PHYS/[rpm] = 1.25 * INT - 2.0                         */
```

6.3.29 COMPARISON_QUANTITY

Prototype:

COMPARISON_QUANTITY Name

Parameters:

ident Name:	Unique identifier in the program (Reference to a valid MEASUREMENT)

Description:

This keyword is references a valid MEASUREMENT in the ASAP2 file. Semantic Interpretation (for a CURVE, a CHARACTERISTIC with only one AXIS_DESC) : The conventional workpoint for a -CURVE has only one input quantity (assigned to AXIS_DESCR) and moves on the CURVE. The 'free-moving' workpoint in an xy diagram of a CURVE is described by two quantities (the conventional input quantity with the AXIS_DESC, the x-axis, and an additional comparison quantity described as an optional attribute directly with the CURVE, the y-axis).The 'free-moving' workpoint does not move on the CURVE, but on the xy-diagram in which the CURVE is located. The crossing of the free-moving workpoint and the CURVE would describe an EVENT. Such display is required by calibration engineers of automatic transmission control (EVENT=gear shift). When this keyword with a CURVE is present, the workpoint display of the MCD system shall apply the INPUT_QUANTITY and the COMPARISON_QUANTITY in the xy-diagram.

6.3.30 COMPU_METHOD

Prototype:

```
/begin COMPU_METHOD      Name LongIdentifier ConversionType Format Unit
                         [-> FORMULA]
                         [-> COEFFS]
                         [-> COMPU_TAB_REF]
/end COMPU_METHOD
```

Parameters:

ident Name: identifier in the program for the conversion method

string LongIdentifier: comment, description

enum ConversionType: possible types:

- TAB_INTP: table with interpolation
- TAB_NOINTP: table without interpolation
- TAB_VERB: verbal conversion table
- RAT_FUNC: fractional rational function of the following type
  $f(x)=(axx + bx + c)/(dxx + ex + f)$
  for which:
  INT = f(PHYS)
  Coefficients a, b, c, d, e, f are specified by the optional COEFFS keyword.
  Important: For these coefficients restrictions have to be defined because this general equation cannot always be inverted.
- FORM: conversion based on the formula specified by the optional FORMULA keyword.

string Format: display format in %[length].[layout]; length indicates the overall length; layout indicates the decimal places. The format string should never be empty as "".

string Unit: physical unit

Optional parameters:

-> FORMULA: Formula to be used for the conversion

-> COEFFS: This keyword is used to specify coefficients a, b, c, d, e, f for the fractional rational function of the following type
$(axx + bx + c) / (dxx + ex + f)$

-> COMPU_TAB_REF: This keyword is used to specify a conversion table (reference to COMPU_TAB data record).

COMPU_METHOD

Description:

Specification of a conversion method

Example:

```
/begin COMPU_METHOD      TMPCON1          /* name */
                         "conversion method for engine temperature"
                         TAB_NOINTP       /* convers_type */
                         "%4.2"           /* display format */
                         "°C"             /* physical unit */
    COMPU_TAB_REF        MOTEMP1
/end COMPU_METHOD

/begin COMPU_METHOD      TMPCON2          /* name */
                         "conversion method for air temperature"
                         FORM             /* convers_type */
                         "%4.2"           /* display format */
                         "°C"             /* physical unit */
    /begin FORMULA       "3*X1/100 + 22.7"   /end FORMULA
/end COMPU_METHOD
```

6.3.31 COMPU_TAB

Prototype:

```
/begin COMPU_TAB           Name LongIdentifier ConversionType NumberValuePairs
                           { InVal OutVal }*
                           [->DEFAULT_VALUE]
/end COMPU_TAB
```

Parameters:

ident Name: identifier in the program for the conversion table
string LongIdentifier: comment, description
enum ConversionType: following types are possible:
TAB_INTP: table with interpolation
TAB_NOINTP: table without interpolation
Note: This parameter is a redundant information because the record defined with COMPU_METHOD contain it too. Therefore this parameter is going to be canceled at ASAP2 version 2.00.
int NumberValuePairs: number of successive value pairs for this conversion table
long InVal: axis point
float OutVal: axis value

Optional parameters

-> DEFAULT_VALUE: string used as OutVal for display when the ECU value is out of any declared range. This string shall not be selectable for calibration (when writing to the ECU).

Description:

Conversion table for conversions that cannot be represented as a function.

Example:

```
/begin COMPU_TAB           TT                    /* name */
                           "conversion table for oil temperatures"
                           TAB_NOINTP            /* convers_type */
                           7                     /* number_value_pairs */
                           1  4.3  2  4.7  3  5.8  4  14.2
                           5  16.8  6  17.2  7  19.4     /* value pairs */
/end COMPU_TAB
```

6.3.32 COMPU_TAB_REF

Prototype:

COMPU_TAB_REF	ConversionTable

Parameters:

ident ConversionTable: reference to the data record which contains the conversion table (see COMPU_TAB).

Description:

Reference to the data record which contains the conversion table (see keyword COMPU_TAB).
Note: COMPU_TAB_REF may only refer to objects of type COMPU_TAB or COMPU_VTAB.

Example:

```
COMPU_TAB_REF              TEMP_TAB          /* TEMP_TAB: conversion table */
```

6.3.33 COMPU_VTAB

Prototype:

```
/begin COMPU_VTAB          Name LongIdentifier ConversionType NumberValuePairs
                           { InVal OutVal }*
                           [->DEFAULT_VALUE]
/end COMPU_VTAB
```

Parameters:

ident Name: identifier in the program for the verbal conversion table

string LongIdentifier: comment, description

enum ConversionType: at present only the following types are possible:
TAB_VERB: verbal conversion table
Note: This parameter is a redundant information because the record defined with COMPU_METHOD contain it too. Therefore this parameter is going to be canceled at ASAP2 version 2.00.

int NumberValuePairs: number of successive value pairs for this conversion table

long InVal: byte value

string OutVal: description (meaning) of the corresponding byte value

Optional parameters

-> DEFAULT_VALUE: string used as OutVal for display when the ECU value is out of any declared range. This string shall not be selectable for calibration (when writing to the ECU).

Description:

Conversion table for the visualisation of bit patterns

Example:

```
/begin COMPU_VTAB          TT                 /* name */
                           "engine status conversion"
                           TAB_VERB           /* convers_type */
                           4                  /* number_value_pairs */
                           0 "engine off"     /* value pairs */
                           1 "idling"
                           2 "partial load"
                           3 "full load"
/end COMPU_VTAB
```

6.3.34 COMPU_VTAB_RANGE

Prototype:

/begin COMPU_VTAB_RANGE
Name LongIdentifier ***NumberValueTriples***
{ InValMin InValMax OutVal }*
[->DEFAULT_VALUE]
/end COMPU_VTAB_RANGE

Parameters:

ident Name: identifier in the program for the verbal range based conversion table

string LongIdentifier: comment, description

int NumberValueTriples: number of successive value triples for this verbal range based conversion table

float InValMin: lower limit as float value, needs to be integer ECU value when assigned to “non-float” definitions.

float InValMax: upper limit as float value, needs to be integer ECU value when assigned to “non-float” definitions.

string OutVal: display string for the value range

Optional parameters

-> DEFAULT_VALUE: string used as OutVal for display when the ECU value is out of any declared range. This string shall not be selectable for calibration (when writing to the ECU).

Description:

Conversion table for the assignment of display strings to a value range. In particular this is useful for ASAP2 definitions with the data type ‘floating point’ (referred as FLOAT definitions).

For FLOAT definitions, the declared string is displayed for InValMin <= ECU value < InValMax, with InValMin, InValMax as floating point values.

For non-FLOAT definions, the declared string is displayed for InValMin <= ECU value <= InValMax, with InValMin, InVal as integer values.

Note : InValMin and InValMax can have the same value to express an assignement of one ECU value to a string (as in COMPU_VTAB); this is not realistic for floating point (and therefore not supported).

Note : Overlapping ranges may not be declared. The ASAP2 file is invalid in case of overlapping ranges within COMPU_VTAB_RANGE. But still, the upper limit of one range may be the same FLOAT value than the lower limit of the following range in case of a FLOAT definition (see display rules).

Note : When a COMPU_METHOD with COMPU_VTAB_RANGE is
used for calibration (writing of values to ECU), the InValMin is used when the assigned STRING(OutVal) is selected in the user interface.

Note : If the optional DEFAULT_VALUE is declared, this string is displayed when the ECU value is out of any declared range. This string shall not be selectable for calibration.

Example:

```
/begin COMPU_VTAB_RANGE
        TT                          /* name */
        “engine status conversion”
        5
        0 0    “ONE“
        1 2    “first_section“
        3 3    “THIRD“
        4 5    “second_section“
        6 500  “usual_case“
        DEFAULT_VALUE “Value_out_of_Range“
/end COMPU_VTAB_RANGE
```

6.3.35 CPU_TYPE

Prototype:

CPU_TYPE	CPU

Parameters:

string CPU:	CPU identifier

Description:

CPU identification

Example:

```
CPU_TYPE                    "INTEL 4711"
```

6.3.36 CUSTOMER

Prototype:

CUSTOMER	Customer

Parameters:

string Customer:	customer name

Description:

This keyword allows a customer name to be specified.

Example:

CUSTOMER	"LANZ - Landmaschinen"

6.3.37 CUSTOMER_NO

Prototype:

CUSTOMER_NO	Number

Parameters:

string Number:	customer number

Description:

Customer number as string.

Example:

```
CUSTOMER_NO                "191188"
```

6.3.38 DATA_SIZE

Prototype:

DATA_SIZE	Size

Parameters:

int Size:	data size in bits

Description:

Data size in bits

Example:

DATA_SIZE	16

6.3.39 DEF_CHARACTERISTIC

Prototype:

```
/begin DEF_CHARACTERISTIC                    { Identifier } *
/end DEF_CHARACTERISTIC
```

Parameters:

ident Identifier: Identifier of those adjustable objects that are defined in respective function.

Description:

This keyword can be used to declare some adjustable objects to be defined in respective function (function orientation).
Note: DEF_CHARACTERISTIC may only refer to objects of type AXIS_PTS or CHARACTERISTIC.

Example:

```
/begin DEF_CHARACTERISTIC        INJECTION_CURVE  DELAY_FACTOR
/end DEF_CHARACTERISTIC
```

6.3.40 DEFAULT_VALUE

Prototype:

DEFAULT_VALUE   display_string

Parameters:

string: display_string

Description:

Optional String which can be applied with COMPU_TAB, COMPU_VTAB and COMPU_VTAB_RANGE, used as OutVal for display when the ECU value is out of any declared range. This string shall not be selectable for calibration (when writing to the ECU).

Example:

DEFAULT_VALUE “overflow_state“

6.3.41 DEPENDENT_CHARACTERISTIC

Prototype:

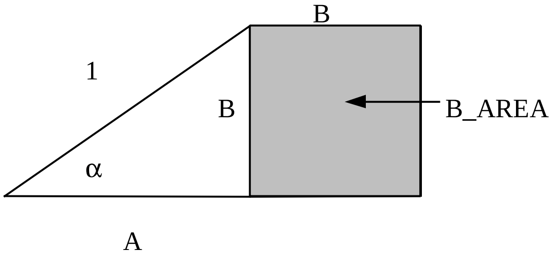
/begin DEPENDENT_CHARACTERISTIC Formula (Characteristic)*
/end DEPENDENT_CHARACTERISTIC

Parameters:

string Formula:	Formula to be used for the calculation of characteristic from the value of other characteristics
ident Characteristic:	Identifier of those adjustable objects that are used for the calculation of this characteristic.

Description:

This keyword allows dependent characteristics to be specified. For this, other characteristics can be combined into one characteristic whose consistent value is automatically derived by the application system. Upon adjusting one of the characteristics, this characteristic is then also automatically adjusted according to the chosen formula (see also VIRTUAL_CHARACTERISTIC). Consider for example a rectangular triangle with a hypothenuse of length 1,



where the length of the other sides are the characteristics A and B. When adjusting A the characteristic B has to be adjusted accordingly to B = sqrt (1- A*A). The relation between the involved characteristics is described on the physical level. Also other characteristic might depend on B, e.g. B_AREA = B * B. A dependent characteristic should not be adjustable by itself, but only through the adjustment of a characteristic it depends on.

The following example makes clear how the calibration process takes place. Assume for each of the characteristics A, B, and B_AREA a conversion formula of *hex = f(phys) = 100 * phys* and assume that the value A_{hex} is 60 (decimal). Then A_{phys} = A_{hex} / 100 = 0.6. According to the formula B = sqrt (1- A*A), B_{phys} = 0.8 and B_{hex} = B_{phys} * 100 = 80 (decimal). According to B_AREA = B*B, we have B_AREA_{phys} = 0.64 and therefore B_AREA_{hex} = 64 (decimal).

Used in CHARACTERISTIC.

Example:

```
/begin DEPENDENT_CHARACTERISTIC
        „sqrt(1-X1*X1)“
        A
/end DEPENDENT_CHARACTERISTIC
```

6.3.42 DEPOSIT

Prototype:

DEPOSIT	Mode

Parameters:

enum Mode: Deposit of the axis points of a characteristic curve or map:

ABSOLUTE:	absolute axis points
DIFFERENCE:	difference axis points

Description:

The axis points of a characteristic can be deposited in two different ways in the memory:
a) The individual axis point values are deposited as absolute values.
b) The individual axis points are deposited as differences. Each axis point value is determined on the basis of the adjacent axis point (predecessor) and the corresponding difference. As reference point for the first axis point <maxvalue> is used:

1-byte-size: <maxvalue> = 2^8 (256)
2-byte-size: <maxvalue> = 2^{16} (65536)
4-byte-size: <maxvalue> = 2^{32}

Example:

DEPOSIT	DIFFERENCE

6.3.43 DISPLAY_IDENTIFIER

Prototype:

DISPLAY_IDENTIFIER display_name

Parameters:

ident: display_name

Description:

This identifier can be used as a alternative name in the Measurement and Calibration System. DISPLAY_IDENTIFIERs can constitute an alternative set of names.

NOTE : The display_name does not have to be unique and is not referenced elsewhere. But is recommended that the display identifier shall be unique in order to avoid confusion in the user interface of the MCD system.

Example:

DISPLAY_IDENTIFIER load_engine

6.3.44 DIST_OP_X/_Y/_Z

Prototype:

DIST_OP_X/_Y/_Z	Position Datatype

Parameters:

int Position:	Position of the distance operand in the deposit structure.
datatype Datatype:	Data type of the distance operand.

Description:

Description of the distance operand in the deposit structure to compute the axis points for fixed characteristic curves and fixed characteristic maps (see also keyword FIX_AXIS_PAR_DIST). The axis points distribution for fixed characteristic curves or fixed characteristic maps is derived from the two 'offset' and 'distance' parameters as follows:

X_i = Offset + (i - 1)*Distance   i = { 1...numberofaxispts }

or

Y_k = Offset + (k - 1)* Distance   k = { 1...numberofaxispts }

or

Z_m = Offset + (m - 1)* Distance   m = { 1...numberofaxispts }

Example:

DIST_OP_X	21 UWORD

6.3.45 ECU

Prototype:

ECU	ControlUnit

Parameters:

string ControlUnit:	control unit identifier

Description:

String for identification of the control unit.

Example:

ECU "Steering control"

6.3.46 ECU_ADDRESS

Prototype:

ECU_ADDRESS	Address

Parameters:

long Address	Address of the measurement in the memory of the control unit.

Description:

ECU_ADDRESS is used to describe the address of an measurement. It should replace the specific IF_DATA (IF_DATA ASAP1B_ADDRESS). It can be used in MEASUREMENT only.

Example:

```
ECU_ADDRESS          0x12FE
```

6.3.47 ECU_CALIBRATION_OFFSET

Prototype:

```
ECU_CALIBRATION_OFFSET            Offset
```

Parameters:

long Offset	Offset that has to be added to each address of a characteristic

Description:

ECU_CALIBRATION_OFFSET is used to describe a fixed address offset when accessing characteristics in the control unit due to

- near pointers in calibration objects. Some record layouts include near pointers inside a calibration objects from which the calibration system has to compute the absolute values by adding the ECU_CALIBRATION_OFFSET (CDAMOS)
- variant coding. Some ECU projects include multiple data sets for different engine or vehicle projects served by one common ECU. By using the ECU_CALIBRATION_OFFSET, a selection for project base address can be made

Used in MOD_PAR.

Example:

```
ECU_CALIBRATION_OFFSET            0x1000
```

6.3.48 EPK

Prototype:

EPK	Identifier

Parameters:

string Identifier:	EPROM identifier

Description:

EPROM identifier string.

Example:

```
EPK                          "EPROM identifier test"
```

6.3.49 ERROR_MASK

Prototype:

ERROR_MASK Mask

Parameters:

long Mask:	mask to mask out selected bits

Description:

The ERROR_MASK keyword can be used to mask bits of a MEASUREMENT which indicate that the value is in error. The Measurement and Calibration System may apply this mask to display the error status of a measurement value. The error mask is usually a single bit; separate measurements should be defined in situations where each bit indicates a different type of error.

Example:

```
ERRROR_MASK 0x00000001
```

6.3.50 EVENT_GROUP

Prototype:

```
/begin EVENT_GROUP          RasterGrpName  ShortName
                            {RasterID}*
/end  EVENT_GROUP
```

Parameters:

string RasterGrpName:	Name of the Group of Event Channels (name for one sample rate of an ECU supported data acquisition mechanism, i.e. the ASAP1a-CCP)
string ShortName:	Short Display Name of the Event Channel Name Recommendation : The string length shall not exceed 9 characters
int RasterID:	Event Channel No., references to the data acquisition event channels of the ECU which are a member of this group.

Description:

Each available „service“ can be described by one ECU_DAQ_EVENT within IF_DATA(Module) and then can be referenced within the ASAP1b-QP_BLOB statement of the SOURCE keyword.

Note : This keyword is only used in the context of the IF_DATA(MODULE) applied for the ASAP1a-CCP.

Example:

```
/begin EVENT_GROUP          "Group of Events" "G1"  2 5 8 9
/end  EVENT_GROUP
```

6.3.51 EXTENDED_LIMITS

Prototype:

EXTENDED_LIMITS LowerLimit UpperLimit

Parameters:

float LowerLimit:	extended range of table values, lower limit
float UpperLimit:	extended range of table values, upper limit

Description:

This keyword can be used to specify an extended range of values. In the application system, for example, when leaving the standard range of values (mandatory parameters 'lower limit' and 'upper limit' in the CHARACTERISTIC data record) a warning could be generated (extended limits enabled only for "power user")

Example:

```
EXTENDED_LIMITS          0 6000.0
```

6.3.52 FIX_AXIS_PAR

Prototype:

FIX_AXIS_PAR Offset Shift Numberapo

Parameters:

int Offset:	'offset' parameter to calculate the axis points of fixed characteristic curves or maps (see description).
int Shift:	'shift' parameter to calculate the axis points of fixed characteristic curves or maps (see description).
int Numberapo:	number of axis points

Description:

Typical of fixed characteristic curves and fixed characteristic maps is that, in contrast with standard and group characteristics, the axis points are not deposited individually in the program data of the ECU program but are derived from the two parameters 'offset' and 'shift'. In the current deposit methods both parameters are contained in the description file. In future deposit methods both methods could well be part of the deposit structure of the adjustable objects.
The axis points of fixed characteristic curves or maps are calculated as follows:

$X_i = \text{Offset} + (i - 1)*2^{\text{Shift}} \qquad i = \{ 1...\text{numberapo} \}$

or

$Y_k = \text{Offset} + (k - 1)*2^{\text{Shift}} \qquad k = \{ 1...\text{numberapo} \}$

or

$Z_m = \text{Offset} + (m - 1)*2^{\text{Shift}} \qquad m = \{ 1...\text{numberapo} \}$

Remark:
This keyword is equivalent to FIX_AXIS_PAR_DIST but differs in parameter ‘Shift’ (see FIX_AXIS_PAR_DIST).

Example:

```
FIX_AXIS_PAR                0 32 8
```

6.3.53 FIX_AXIS_PAR_DIST

Prototype:

FIX_AXIS_PAR_DIST Offset Distance Numberapo

Parameters:

int Offset:	'offset' parameter to calculate the axis points of fixed characteristic curves or maps (see description).
int Distance:	'distance' parameter to calculate the axis points of fixed characteristic curves or maps (see description).
int Numberapo:	number of axis points

Description:

Typical of fixed characteristic curves and fixed characteristic maps is that, in contrast with standard and group characteristics, the axis points are not deposited individually in the program data of the ECU program but are derived from the two parameters 'offset' and 'distance'. In the current deposit methods both parameters are contained in the description file. In future deposit methods both methods could well be part of the deposit structure of the adjustable objects.
The axis points of fixed characteristic curves or maps are calculated as follows:

X_i = Offset + (i - 1)*Distance i = { 1...numberapo }

or

Y_k = Offset + (k - 1)*Distance k = { 1...numberapo }

or

Z_m = Offset + (m - 1)*Distance m = { 1...numberapo }

Remark:
This keyword is equivalent to FIX_AXIS_PAR but differs in parameter ‘Distance’ (see FIX_AXIS_PAR).

Example:

FIX_AXIS_PAR_DIST 0 100 8

6.3.54 FIX_AXIS_PAR_LIST

Prototype:

```
/begin FIX_AXIS_PAR_LIST
        { AxisPts_Value }*
/end FIX_AXIS_PAR_LIST
```

Parameters:

float AxisPts_Value — List of "ECU-Original" Values as implied by the ECU algorithm. The number of values must match with the MaxAxisPoints attribute of the AXIS_DESCR referencing FIX_AXIS_PAR_LIST. The COMPU_METHOD assigned to the AXIS_DESCR shall be applied to achieve the actual display values.
NOTE : The data type shall be integer in case of an assignment to a non-float definition).

Description:

Allows the description of any value combination of a virtual axis (FIX_AXIS, axis points not in the ECU memory). Other methods (FIX_AXIS_PAR, FIX_AXIS_PAR_DIST) implicitely assume an interpolation algorithm in the ECU. But axis descriptions are also used e.g. to span status tables.
The values are the input for the COMPU_METHOD assigned to the axis. Even a verbal table could be applied as COMPU_METHOD (i.e for the axis description of status tables on which no interpolation is applied).

Example:

```
/begin FIX_AXIS_PAR_LIST          2 5 9   /end FIX_AXIS_PAR_LIST
```

6.3.55 FIX_NO_AXIS_PTS_X/_Y/_Z

Prototype:

FIX_NO_AXIS_PTS_X/_Y/_Z NumberOfAxisPoints

Parameters:

int NumberOfAxisPoints: Dimensioning of characteristic curves or characteristic maps with a fixed number of axis points

Description:

This keyword indicates that all characteristic curves or characteristic maps are allocated a fixed number of X-axis and Y-axis points. In a RECORD_LAYOUT data record, this keyword cannot be used simultaneously with the keyword NO_AXIS_PTS_X (for FIX_NO_AXIS_PTS_X) or NO_AXIS_PTS_Y (for FIX_NO_AXIS_PTS_Y)

Example:

```
FIX_NO_AXIS_PTS_X           17
```

6.3.56 FNC_VALUES

Prototype:

FNC_VALUES Position Datatype IndexMode Addresstype

Parameters:

int Position: position of table values (function values) in the deposit structure (description of sequence of elements in the data record).

datatype DataType: data type of the table values

enum IndexMode: for characteristic maps, this attribute is used to describe how the 2-dimensional table values are mapped onto the 1-dimensional address space:

COLUMN_DIR	deposited in columns
ROW_DIR	deposited in rows

Both concepts 'columns' and 'rows' relate to the XY coordinate system (see also Appendix B: Record layouts).

For characteristic cuboids each XY plane is mapped as above. The cuboid is stored as an array of maps with incremental Z coordinates.

<u>Additional indexMode (since version 1.2):</u>

ALTERNATE_WITH_X

maps: deposited in columns, the columns of table values alternate with the respective X-coordinates.

curves: table values and X-coordinate values are deposited alternating.

ALTERNATE_WITH_Y

maps: deposited in rows, the rows of table values alternate with the respective Y-coordinates (maps only).

ALTERNATE_CURVES

curves: curves which share a common axis are deposited in columns; each row of memory contains values for all the shared axis curves at a given axis breakpoint. Required in order to represent characteristics which correspond to arrays of structures in ECU program code. In the example code below, DT10, DT20, etc are treated as separate curves which may have different conversions or limits:-

```
typedef struct    {
            int DT10;
            int DT20;
            int DT30;
```

FNC_VALUES

```
     int DT40;
} VXP_TYPE;

const VXP_TYPE VX_PLUS_DELAY_TIMES[5] = {
            { 10, 3, 4, 8 },
            { 12, 2, 4, 6 },
            { 17, 9, 5, 8 },
            { 10, 1, 4, 8 },
            { 18, 3, 8, 8 },
       };
```

addrtype Addresstype: addressing of the table values (see enum addrtype).

Description:

Description of the table values (function values) of an adjustable object

Example:

FNC_VALUES 7 SWORD COLUMN_DIR DIRECT

6.3.57 FORMAT

Prototype:

FORMAT	FormatString

Parameters:

string FormatString:	display format in %[length].[layout]; length indicates the overall length; layout indicates the decimal places

Description:

This keyword allows a special display format to be specified for some MEASUREMENT, CHARACTERISTIC or AXIS_PTS object. If exists this display format is used instead of display format specified in respective COMPU_METHOD data record. The format string should never be empty as "".

Example:

FORMAT	"%4.2"

6.3.58 FORMULA

Prototype:

/begin FORMULA f(x)
[-> FORMULA_INV]
/end FORMULA

Parameters:

string f(x): function to calculate the physical value from the hexadecimal, control unit internal value. The interpretation proceeds from left to right. Operator preferences, such as power before product/quotient before sum/difference, are taken into account. Brackets are allowed. The following operation symbols can be used:

Basic operations:

+	for sums
-	for differences
*	for products
/	for quotients
^	for powers

Logical operators: interpretation from left to right

&	logical AND
½	logical OR
>>	shift right
<<	shift left
XOR	exclusive OR
~	logical NOT

Trigonometric functions:
sin(x), con(x), tan(x)
arcsin(x), arccos (x), arctan (x)
sinh(x), cosh(x), tanh(x)

Exponential function:
exp(x)

Logarithmic functions:
ln(x)
log(x)

Square root, absolute amount:
sqrt(x)
abs(x)

FORMULA

Optional parameters:

-> FORMULA_INV: function to calculate the hexadecimal, control unit internal value from the physical value. This parameter is mandatory in formulas used for the conversion of adjustable objects. It is optional only for measurement objects.

Note:

Certain functions in the application system can only be used for those measurement objects for which this parameter is specified (e.g. scalable DAC output, triggering).

Description:

This keyword allows any kind of formula to be specified for the conversion of measurement values, axis points or table values of an adjustable object from their hexadecimal (ECU internal) format into the physical format. The interpretation of the formula must be supported by a formula interpreter in the operating system.

Example:

```
/begin FORMULA          "sqrt( 3 - 4*(sin(X1))^2 )"
/end FORMULA
```

6.3.59 FORMULA_INV

Prototype:

FORMULA_INV	g(x)

Parameters:

string g(x): function for calculation of the hexadecimal, control unit internal value from the physical value. The interpretation proceeds from left to right. Operator preferences, such as power before product/quotient before sum/differenc, are taken into account. Brackets are allowed. Permissible operation symbols: see keyword FORMULA, page 100.

Description:

This keyword allows any kind of formula to be specified for the conversion of measurement values, axis points or table values of an adjustable object from their physical format into the hexadecimal (ECU internal) format. The interpretation of the formula must be supported by a formula interpreter in the operating system.

Example:

Inversion function e.g. for keyword FORMULA (see page 100)

```
FORMULA_INV              "arcsin( sqrt( (3 - (X1)^2)/4 ) )"
```

6.3.60 FRAME

Prototype:

```
/begin FRAME                Name
                              LongIdentifier
                              ScalingUnit
                              Rate
                            [-> FRAME_MEASUREMENT]
                            {-> IF_DATA} *

/end FRAME
```

Parameters:

ident Name:	Identifier in the program, referencing is based on this 'name'
string LongIdentifier:	comment, description
int ScalingUnit:	This parameter defines the basic scaling unit. The following parameter 'Rate' relates on this scaling unit. The value of ScalingUnit is coded as shown in ‘Table 4: Codes for scaling units (CSE)’ (page130).
long Rate:	The maximum refresh rate of the concerning measurement source in the control unit. The unit is defined with parameter 'ScalingUnit'.

Optional parameters:

-> FRAME_MEASUREMENT:

Use this keyword to define the frames measurementobjects.

-> IF_DATA: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitRead(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

Description:

For the structuring of a car network involving a very large number of measuring channels, function frames can be defined.

These function frames shall be used in the application system to allow the selection lists for the selection of measuring channels to be represented in a structured manner on the basis of functional viewpoints (function orientation).

This will also be used to describe the packaging of measurment data into sources for CAN frames in a network environment.

Example:

```
/begin FRAME                      ABS_ADJUSTM
                                  "function group ABS adjustment"
                                    3
                                    2      /* 2 msec. */
        FRAME_MEASUREMENT         LOOP_COUNTER  TEMPORARY_1
/end FRAME
```

6.3.61 FRAME_MEASUREMENT

Prototype:

FRAME_MEASUREMENT { Identifier } *

Parameters:

ident Identifier:	Identifier of quantity of respective FRAME (reference to measurement object).

Description:

This keyword can be used to define quantities of respective FRAME.

Example:

```
FRAME_MEASUREMENT       WHEEL_REVOLUTIONS  ENGINE_SPEED
```

6.3.62 FUNCTION

Prototype:

```
/begin FUNCTION          Name LongIdentifier
                         {-> ANNOTATION} *
                         [-> DEF_CHARACTERISTIC]
                         [-> REF_CHARACTERISTIC]
                         [-> IN_MEASUREMENT]
                         [-> OUT_MEASUREMENT]
                         [-> LOC_MEASUREMENT]
                         [-> SUB_FUNCTION]
/end FUNCTION
```

Parameters:

ident Name: Identifier in the program, referencing is based on this 'name'

string LongIdentifier: comment, description

Optional parameters:

-> ANNOTATION: Set of notes (represented as multi-line ASCII description texts) which are related. Can serve e.g. as application note.

-> DEF_CHARACTERISTIC: This keyword can be used to define those adjustable objects which are **defined** in respective function. *

-> REF_CHARACTERISTIC: If the function contains **references** to some adjustable objects, this keyword can be used to describe this references.

-> IN_MEASUREMENT: Use this keyword to define the input measurement objects of respective function (input variables).

-> OUT_MEASUREMENT: Use this keyword to define the output measurement objects of respective function (output variables).

-> LOC_MEASUREMENT: Use this keyword to define the local measurement objects of respective function (local variables: scope is limited to this function).

-> SUB_FUNCTION: This keyword can be used to describe the function hierarchy. If the respective function is subdivided into subfunctions, use this keyword to define the subfunctions.

Description:

For the structuring of projects involving a very large number of adjustable objects and measuring channels, functions can be defined. These functions shall be used in the application system to allow the selection lists for the selection of adjustable objects and measuring channels to be represented in a structured manner on the basis of functional viewpoints (function orientation).

FUNCTION

Remark: Since ASAP2 version 1.20 the references between functions and measurement objects resp. adjustable objects can be described either with keyword CHARACTERISTIC, AXIS_PTS and MEASUREMENT (see FUNCTION_LIST) or with keyword FUNCTION.

Example:

```
/begin FUNCTION                 ID_ADJUSTM                         /* name */
                                "function group idling adjustment"
    /begin DEF_CHARACTERISTIC   INJECTION_CURVE
    /end DEF_CHARACTERISTIC
    /begin REF_CHARACTERISTIC   FACTOR_1
    /end REF_CHARACTERISTIC
    /begin IN_MEASUREMENT       WHEEL_REVOLUTIONS  ENGINE_SPEED
    /end IN_MEASUREMENT
    /begin OUT_MEASUREMENT      OK_FLAG  SENSOR_FLAG
    /end OUT_MEASUREMENT
    /begin LOC_MEASUREMENT      LOOP_COUNTER  TEMPORARY_1
    /end LOC_MEASUREMENT
    /begin SUB_FUNCTION         ID_ADJUSTM_SUB
    /end SUB_FUNCTION
/end FUNCTION
```

6.3.63 FUNCTION_LIST

Prototype:

```
/begin FUNCTION_LIST        ( Name ) *
/end FUNCTION_LIST
```

Parameters:

ident Name:	list of references to higher-order functions (see FUNCTION)

Description:

This keyword can be used to specify a list of 'functions' to which the relevant adjustable object has been allocated (function orientation).

Remark: Since ASAP2 version 1.20 the keyword FUNCTION comprises some additional features to describe functional structure and dependencies. The keyword FUNCTION_LIST is going to be canceled at ASAP2 version 2.00.

Example:

```
/begin FUNCTION_LIST        ID_ADJUSTM  FL_ADJUSTM  SPEED_LIM
/end FUNCTION_LIST
```

6.3.64 GROUP

Prototype:

```
/begin GROUP
                    GroupName GroupLongIdentifier
                    {-> ANNOTATION} *
                    [-> ROOT]
                    [-> REF_CHARACTERISTIC]
                    [-> REF_MEASUREMENT]
                    [-> FUNCTION_LIST]
                    [-> SUB_GROUP]
/end GROUP
```

Parameters:

ident GroupName: Identifier of the group

string GroupLongIdentifier:
Comment, description of the group within a grouping mechanism.

Optional parameters:

-> ANNOTATION: Set of notes (represented as multi-line ASCII description texts) which are related. Can serve e.g. as application note.

-> ROOT This keyword indicates that the group constitutes an independent grouping mechanism (root level) which the MCD system may use as a root point for the hierarchical presentation of groups. All groups referenced via SUB_GROUP (including nested references) constitute a set of groups belonging to the grouping mechanism.
Examples for such grouping mechanisms : Group Name = {Software_Components, Calibration_Components, Editor_Selection_Lists}

-> REF_CHARACTERISTIC: If the group contains **references** to some adjustable objects, this keyword can be used to describe these references.

-> REF_MEASUREMENT: If the group contains **references** to some measurement objects, this keyword can be used to describe these references.

-> FUNCTION_LIST:	This keyword can be used to specify a list of references to functions.
-> SUB_GROUP:	This keyword can be used to describe the group hierarchy. If the respective group is subdivided into subgroups, use this keyword to define the subgroups. In particular, SUB_GROUP references the groups belonging to a grouping mechanism indicated with the optional keyword ROOT (see above).

Description:

These GROUPs shall be used in the application system to provide selection lists (groups) of adjustable objects and measuring channels.

For the structuring of projects involving a very large number of adjustable objects and measuring channels, an **unlimited number of grouping mechanisms, each constituted from a root group containing subgroups** (including nested references)**,** can be defined. Such root groups are used in the MCD system for initial display of the available groups, as the root of a tree containing the referenced subgroups. Use cases are e.g. software_components which define the C file assignment, calibration_components which describe the calibration engineer's viewpoint, editor_selection_lists which can define the presentation of calibration objects and their related measurement quantities.

Example:

```
/begin GROUP                    SOFTWARE_COMPONENTS
        "assignment of the definitions to C files"
        ROOT
        /begin SUB_GROUP   INJE   C6TD
        /end SUB_GROUP
/end GROUP

/begin GROUP  INJE "Subsystem Injection"
        /begin SUB_GROUP             injec1 injec2
        /end SUB_GROUP
/end GROUP

/begin GROUP  Injec1 "Module filename Injec1"
        /begin REF_CHARACTERISTIC             INJECTION_CURVE
        /end REF_CHARACTERISTIC
        /begin REF_MEASUREMENT    LOOP_COUNTER  TEMPORARY_1
        /end REF_MEASUREMENT
/end GROUP

/begin GROUP  Injec2 "Module filename Injec2"
        /begin REF_CHARACTERISTIC             INJECTION_ADJUST
        /end REF_CHARACTERISTIC
        /begin REF_MEASUREMENT  GAS_INPUT  WHEEL_SPEED
        /end REF_MEASUREMENT
/end GROUP


/begin GROUP  C6TD “Shift Point Control”
        /begin   SUB_GROUP              c6tdvder c6tdertf
        /end  SUB_GROUP
/end GROUP

/begin GROUP c6tdvder “Module filename c6tdvder”
        /begin  REF_CHARACTERISTIC                SHIFT23_CURVE
        /end  REF_CHARACTERISTIC
        /begin REF_MEASUREMENT    LOOP_COUN2  NO_GEAR
        /end REF_MEASUREMENT
/end GROUP

/begin GROUP c6tderft “Module filename c6tderft”
        /begin REF_CHARACTERISTIC                 LUP23_CURVE
        /end  REF_CHARACTERISTIC
        /begin REF_MEASUREMENT   TRANSMISSION_SP  ENGINE_SPEED
        /end REF_MEASUREMENT
/end GROUP


/begin GROUP                               CALIBRATION_COMPONENTS
        "assignment of the definitions to calibration components"
        ROOT
        /begin SUB_GROUP
          Winter_Test
          Summer_Test
        /end SUB_GROUP
/end GROUP

/begin GROUP  Winter_Test “Flash this in winter time”
        /begin REF_CHARACTERISTIC                 GASOLINE_CURVE
        /end  REF_CHARACTERISTIC
/end GROUP

/begin GROUP Summer_Test “Flash that in summer time”
        /begin REF_CHARACTERISTIC                 SUPER_CURVE
        /end  REF_CHARACTERISTIC
/end GROUP
```

6.3.65 GUARD_RAILS

Prototype:

GUARD_RAILS

Description:

This keyword is used to indicate that an adjustable CURVE, MAP or AXIS_PTS uses guard rails. The Measurement and Calibration System does not allow the user to edit the outermost values or axis points of the adjustable object, but calculates them as follows:-

AXIS_PTS	CURVE	MAP
(X_0) = AXIS_PTS.LowerLimit	$(X_0) = (X_1)$	$(X_i, Y_0) = (X_j, Y_1)$
(X_m) = AXIS_PTS.UpperLimit	$(X_m) = (X_{m-1})$	$(X_i, Y_n) = (X_j, Y_{n-1})$
		$(X_0, Y_j) = (X_1, Y_j)$
		$(X_m, Y_j) = (X_{m-1}, Y_j)$

$0 < i < m$, m = Number of X-axis points
$0 < j < n$, n = Number of Y-axis points

Example:

```
/begin CHARACTERISTIC       F_INJ_CORR        /* name */
                            "Injector correction factor" /* long identifier */
                            CURVE             /* type */
                            0x7140            /* address */
                            REC12             /* deposit */
                            10.0              /* maxdiff */
                            C_INJF            /* conversion */
                            0.0 199.0         /* lower limit, upper limit */
                            GUARD_RAILS       /* uses guard rails */
        /begin AXIS_DESCR                     /* description of X-axis points */
                            STD_AXIS          /* standard axis points */
                            N                 /* reference to input quantity */
                            C_TEMP            /* conversion */
                            10                /* maximum number of axis points */
                            -40.0 150.0       /* lower limit, upper limit */
        /end AXIS_DESCR
/end CHARACTERISTIC
```

6.3.66 HEADER

Prototype:

```
/begin HEADER                Comment
                             [-> VERSION]
                             [-> PROJECT_NO]
/end HEADER
```

Parameters:

string Comment:	comment, description

Optional parameters:

-> VERSION:	version number
-> PROJECT_NO:	project number

Description:

Header information on a project. A project can comprise several ASAP devices.

Example:

```
/begin HEADER                "see also specificatio XYZ of 01.02.1994"
    VERSION                  "BG5.0815"
    PROJECT_NO               M4711Z1
/end HEADER
```

6.3.67 IDENTIFICATION

Prototype:

IDENTIFICATION	Position Datatype

Parameters:

int Position:	position of the 'identifier' in the deposit structure.
datatype Datatype:	word length of the 'identifier''

Description:

Description of an 'identifier' in an adjustable object (see BOSCH: C-DAMOS deposit).

Example:

```
IDENTIFICATION          1 UWORD
```

6.3.68 IF_DATA (AXIS_PTS, CHARACTERISTIC, MEMORY_LAYOUT)

Prototype:

```
/begin IF_DATA           Name
                         [-> DP_BLOB Dp_Data]
                         [-> PA_BLOB Pa_Data]
/end IF_DATA
```

Parameters:

ident Name: identifier of ASAP device. This string must be defined in the A2ML-file. See chapter 8.2 (Designing A2ML-file) for details.

Optional parameters:

-> DP_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitAccess(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

-> PA_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command Access(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

Description:

Definition of interface-specific description data (topic keyword AXIS_PTS, CHARACTERISTIC and MEMORY_LAYOUT).

Example:

```
/begin IF_DATA
    ASAP1B_CCP           /* Name of ASAP device */
    DP_BLOB              /* interface-specific parameters described in A2ML */
        0x12129977 0xFF
    PA_BLOB              /* interface-specific parameters described in A2ML */
        "Pumpenkennfeld" 1  2   17
/end IF_DATA
```

6.3.69 IF_DATA (MEASUREMENT)

Prototype:

```
/begin IF_DATA          Name
                        [-> KP_BLOB KP_Data]
                        [-> DP_BLOB Dp_Data]
                        [-> PA_BLOB Pa_Data]
/end IF_DATA
```

Parameters:

ident Name: identifier of ASAP device. This string must be defined in the A2ML-file. See chapter 8.2 (Designing A2ML-file) for details.

Optional parameters:

-> KP_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitRead(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

-> DP_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitAccess(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.
Note: This keyword is required only if MCD wants to 'write' to the corresponding measurement object (see keyword READ_WRITE of topic MEASUREMENT).

-> PA_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command Access(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.
Note: This keyword is required only if MCD wants to 'write' to the corresponding measurement object (see keyword READ_WRITE of topic MEASUREMENT).

Description:

Definition of interface-specific description data (topic keyword MEASUREMENT).

Example:

```
/begin IF_DATA
        ASAP1B_CCP                  /* Name of ASAP device */
        KP_BLOB 0x12FE  17  3       /* interface-specific parameters described in A2ML */
        DP_BLOB 0x121277 0xFF       /* interface-specific parameters described in A2ML */
        PA_BLOB                     /* interface-specific parameters described in A2ML */
             "Pumpenkennfeld"  1  2   17
/end IF_DATA
```

Minimal Requirements for Data Acquisition:

The interface-specific parameters of measurement objects can be described using keyword IF_DATA.
On the one hand this keyword is optional, i.e. the description file is ASAP2-conform even if this keyword is missing. On the other hand in real application access to any measurement object is only possible if a suitable IF_DATA record is available.
Recommendation: If the description file doesn’t include any interface-specific parameters corresponding to real application system, description file should either contain a record to describe an emulator address (IF_DATA ASAP1B_ADDRESS) or description parameters to acquire the measurement data from CAN-bus (IF_DATA ASAP1B_CAN). Usage of one or both of this IF_DATA-records depends on real case.

6.3.70 IF_DATA ASAP1B_ADDRESS (MEASUREMENT)

Prototype:

```
/begin IF_DATA  ASAP1B_ADDRESS  KP_BLOB Address
/end IF_DATA
```

Parameters:

ASAP1B_ADDRESS:	Identifier of ASAP device.
KP_BLOB:	This keyword marks the interface-specific parameters used at ASAP1b device at call of the command InitRead().
long Address:	Address of measurement object (e.g. emulator RAM address)

Description:

Definition of interface-specific description data (topic keyword MEASUREMENT). This record is recommended if there isn’t any other record available, corresponding to a real application system.

Example:

```
/begin IF_DATA          ASAP1B_ADDRESS        KP_BLOB      0x12FE
/end IF_DATA
```

6.3.71 IF_DATA ASAP1B_CAN (MEASUREMENT)

Prototype:

```
/begin IF_DATA  ASAP1B_CAN
        /begin KP_BLOB          MessageName  Identifier  MessageSize
                                MessageSource   Startbit  DataSize
                                [-> MULTIPLEX]
        /end KP_BLOB
/end IF_DATA
```

Parameters:

ASAP1B_CAN:	Identifier of ASAP device.
KP_BLOB:	This keyword marks the interface-specific parameters used at ASAP1b device at call of the command InitRead().
ident MessageName:	Description of CAN-message.
long Identifier:	CAN-Identifier of that message, the expected measurement object is included in.
int MessageSize:	Size of CAN-message (message data only).
string MessageSource:	Description of message sender.
int Startbit:	The expected measurement data are included in message data at offset ‘Startbit’ (in bits, index of first bit of message data is 0).
int DataSize:	Size of measurement data included in message data.

Optional parameters:

-> MULTIPLEX:	In case of multiplexed message data this keyword can be used to define the ‘mode-signal’ (see MULTIPLEX).

Description:

Definition of interface-specific description data (topic keyword MEASUREMENT). This record is recommended if there isn’t any other record available corresponding to a real application system.

Example:

```
/begin IF_DATA ASAP1B_CAN
        /begin KP_BLOB          „MOTOINFO“  0x123  8  „MOTOR-SG“  5  8
        /end KP_BLOB
/end IF_DATA
```

6.3.72 IF_DATA (FRAME)

Prototype:

```
/begin IF_DATA          Name
                        [-> QP_BLOB QP_Data]
/end IF_DATA
```

Parameters:

ident Name: identifier of ASAP device. This string must be defined in the A2ML-file. See chapter 8.2 (Designing A2ML-file) for details.

Optional parameters:

-> QP_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitRead(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

Description:

Definition of interface-specific description data (topic keyword FRAME).

6.3.73 IF_DATA (MEMORY_SEGMENT)

Prototype:

```
/begin IF_DATA          Name
                        {-> ADDRESS_MAPPING } *
/end IF_DATA
```

Parameters:

ident Name: identifier of ASAP device. This string must be defined in the A2ML-file. See chapter 8.2 (Designing A2ML-file) for details.

Optional parameters:

-> ADDRESS_MAPPING: This keyword can be used to remap the address space of the ECU to an access address.

Description:

Remapping of the address space.

Example:

```
/begin MEMORY_SEGMENT ext_Ram  “external RAM“
  DATA RAM EXTERN 0x30000 0x1000 -1 -1 -1 -1 -1
 /begin IF_DATA ASAP1B_KWP2000
  /* ADDRESS_MAPPING orig_addr      mapping_addr      length */
  ADDRESS_MAPPING     0x4000         0x6000            0x0200
/end IF_DATA
/end MEMORY_SEGMENT
```

6.3.74 IF_DATA (MODULE)

Prototype:

```
/begin IF_DATA          Name
                        {-> SOURCE } *
                        {-> RASTER } *
                        {-> EVENT_GROUP}*
                        [-> SEED_KEY]
                        [-> CHECKSUM]
                        [-> TP_BLOB TP_Data]
/end IF_DATA
```

Parameters:

ident Name: Identifier of ASAP device. This string must be defined in the A2ML-file. See chapter 8.2 (Designing A2ML-file) for details.

Optional parameters:

-> SOURCE: This keyword can be used to describe different 'sources' of measurement, e.g. different acquisition modes (e.g. time synchronous 10 msec, time synchronous 50 msec etc.)

-> RASTER: This keyword can be used to declare an event channel which is part of the SOURCE of data acquisition.
Only used in context of the ASAP 1a-CCP.

-> EVENT_GROUP: This keyword can be used to declare a group of event channels (group of ECU_DAQ_EVENT)
Only used in context of the ASAP 1a-CCP.

-> SEED_KEY: Description of the authentification process. References to dynamic link libraries with a standard API, which contain the authentification algorithms.

-> CHECKSUM: Description of the ECU checksum algoritm. References to dynamic link libraries with a standard API, which contain the checksum algorithm.

-> TP_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitRead(), InitAccess() and Command(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

Description:

Definition of interface-specific description data (topic keyword MODULE).

Example:

IF_DATA (MODULE)

```
/begin IF_DATA
    ASAP1B_CCP              /* Name of ASAP device */
        /begin SOURCE       Time10ms   4   1      /* Source parameter */
        /end SOURCE
        /begin SOURCE       Time50ms   4   5      /* Source parameter */
        /end SOURCE
        TP_BLOB             0xFF 0xFA 0x0A     /* interface-specific parameters */
/end IF_DATA
```

6.3.75 IN_MEASUREMENT

Prototype:

```
/begin IN_MEASUREMENT    { Identifier } *
/end IN_MEASUREMENT
```

Parameters:

ident Identifier: Identifier of input quantity of respective function (reference to measurement object).

Description:

This keyword can be used to define input quantities of respective function.
Note: IN_MEASUREMENT may only refer to objects of type MEASUREMENT.

Example:

```
/begin IN_MEASUREMENT    WHEEL_REVOLUTIONS  ENGINE_SPEED
/end IN_MEASUREMENT
```

6.3.76 LEFT_SHIFT

Prototype:

LEFT_SHIFT	Bitcount

Parameters:

long bitcount:	Shift ‚bitcount‘ bits to the left

Description:

The LEFT_SHIFT keyword is only used within the BIT_OPERATION keyword. See description of BIT_OPERATION.

6.3.77 LOC_MEASUREMENT

Prototype:

/begin LOC_MEASUREMENT { Identifier } *
/end LOC_MEASUREMENT

Parameters:

ident Identifier:	Identifier of local quantity of respective function (reference to measurement object).

Description:

This keyword can be used to define local quantities of respective function.
Note: LOC_MEASUREMENT may only refer to objects of type MEASUREMENT.

Example:

```
/begin LOC_MEASUREMENT                    LOOP_COUNTER  TEMPORARY_1
/end LOC_MEASUREMENT
```

6.3.78 MAP_LIST

Prototype:

/begin MAP_LIST { Name } *
/end MAP_LIST

Parameters:

ident Name: identifier of a MAP (see CHARACTERISTIC)

Description:

This keyword can be used to specify the list of MAPs which comprise a CUBOID. This keyword is required because CUBOID data will not be at contiguous memory locations if a CUBOID is composed of several MAPs.

6.3.79 MAX_GRAD

Prototype:

MAX_GRAD MaxGradient

Parameters:

float MaxGradient: maximum permissible gradient

Description:

This keyword is used to specify a maximum permissible gradient for an adjustable object in relation to an axis:

MaxGrad_x = maximum(absolut(($W_{i,k}$ - $W_{i-1,k}$)/(X_i - X_{i-1})))

MaxGrad_y = maximum(absolut(($W_{i,k}$ - $W_{i,k-1}$)/(Y_i - Y_{k-1})))

Example:

MAX_GRAD 200.0

6.3.80 MAX_REFRESH

Prototype:

MAX_REFRESH ScalingUnit Rate

Parameters:

int ScalingUnit:	this parameter defines the basic scaling unit. The following parameter 'Rate' relates on this scaling unit. The value of ScalingUnit is coded as shown below in ‘Table 4: Codes for scaling units (CSE)’.
long Rate:	the maximum refresh rate of the concerning measurement object in the control unit. The unit is defined with parameter 'ScalingUnit'.

Description:

This optional keyword can be used to specify the maximum refresh rate in the control unit.

Example:

```
MAX_REFRESH         3  15  /* ScalingUnit = 1 msec --> refresh rate = 15 msec */

MAX_REFRESH         998  2  /* ScalingUnit = 998 --> Every second frame */
```

MAX_REFRESH

Code	Unit	Referred to	Comment
0	1 µsec	Time	
1	10 µsec	Time	
2	100 µsec	Time	
3	1 msec	Time	
4	10 msec	Time	
5	100 msec	Time	
6	1 sec	Time	
7	10 sec	Time	
8	1 min	Time	
9	1 hour	Time	
10	1 day	Time	
100	Angular degrees	Angle	
101	Revolutions 360 degrees	Angle	
102	Cycle 720 degrees	Angle	e.g. in case of IC engines
103	Cylinder segment	Combustion	e.g. in case of IC engines
998	When frame available	Event	Source defined in keyword Frame
999	Always if there is new value		Calculation of a new upper range limit after receiving a new partial value, e.g. when calculating a complex trigger condition
1000	Non deterministic		Without fixed scaling

Table 4: Codes for scaling units (CSE)

6.3.81 MEASUREMENT

Prototype:

/begin MEASUREMENT	Name LongIdentifier Datatype Conversion Resolution Accuracy LowerLimit UpperLimit [-> DISPLAY_IDENTIFIER] [-> READ_WRITE] [-> FORMAT] [-> ARRAY_SIZE] [-> BIT_MASK] [-> BIT_OPERATION] [-> BYTE_ORDER] [-> MAX_REFRESH] [-> VIRTUAL] [-> FUNCTION_LIST] [-> ECU_ADDRESS] [-> ERROR_MASK] [-> REF_MEMORY_SEGMENT] {-> ANNOTATION} * {-> IF_DATA } *
/end MEASUREMENT	

Parameters:

ident Name:	unique identifier in the ECU program ***(must be unique within the ASAP2 MODULE)***
string LongIdentifier:	comment, description
datatype Datatype:	data type of the measurement
ident Conversion:	reference to the relevant data record for description of the conversion method (see COMPU_METHOD)
int Resolution:	smallest possible change in bits
float Accuracy:	possible variation from exact value in %
float LowerLimit:	plausible range of table values, lower limit
float UpperLimit:	plausible range of table values, upper limit

Optional parameters

-> DISPLAY_IDENTIFIER:	Can be used as a display name (alternative to the ‘name’ attribute).
-> READ_WRITE:	Keyword to mark this measurement object as 'writeable'.
-> FORMAT:	With deviation from the display format specified with keyword COMPU_TAB referenced by parameter <Conversion> a special display format can be specified to be used to display the measurement values.
-> ARRAY_SIZE:	This keyword marks a measurement object as an array of measurement values.

-> BIT_MASK: With deviation from the standard value 0xFFFFFFFF this parameter can be used to mask out bits.

-> BIT_OPERATION: The BIT_OPERATION keyword can be used to perform operation on the masked out value.

-> BYTE_ORDER: With deviation from the standard value this parameter can be used to specify the byte order (Intel format, Motorola format)

-> MAX_REFRESH: Maximum refresh rate of this measurement in the control unit

-> VIRTUAL: For description of a virtual measurement (see VIRTUAL)

-> ERROR_MASK: With deviation from the standard value 0x00000000 this parameter can be used to mask bits of a MEASUREMENT which indicate that the value is in error.

-> FUNCTION_LIST: This keyword can be used to specify a list of 'functions' to which this measurement object has been allocated.
Remark: Since ASAP2 version 1.20 the keyword FUNCTION comprises some additional features to describe functional structure and dependencies. The keyword FUNCTION_LIST is going to be canceled at ASAP2 version 2.00.

-> IF_DATA: Date record to describe the interface specific description data. The parameters associated with this keyword are described in A2ML by the control unit supplier or the interface module supplier.

-> ECU_ADDRESS: Addess of the measurement in the memory of the control unit.

-> REF_MEMORY_SEGMENT: Reference to the memory segment which is needed if the address is not unique (this occurs in the case of lapping address ranges (overlapping memory segments).

-> ANNOTATION: Set of notes (represented as multi-line ASCII description texts) which are related. Can serve e.g. as application note.

Description:

The MEASUREMENT keyword is used to describe the parameters for the processing of a measurement object.

Example:

```
/begin MEASUREMENT      N               /* name */
                        "Engine speed"  /* long identifier */
                        UWORD           /* datatype */
                        R_SPEED_3       /* conversion */
                        2               /* resolution */
                        2.5             /* accuracy */
                        120.0           /* lower limit */
                        8400.0          /* upper limit */
    BIT_MASK            0x0FFF
    /begin BIT_OPERATION
                         RIGHT_SHIFT  4    /*4 positions*/
                         SIGN_EXTEND
     /end BIT_OPERATION
    BYTE_ORDER                  MSB_FIRST
    REF_MEMORY_SEGMENT Data2
    /begin FUNCTION_LIST        ID_ADJUSTM  FL_ADJUSTM
    /end FUNCTION_LIST
    /begin IF_DATA ISO          SND 0x10 0x00 0x05 0x08 RCV 4  long
    /end IF_DATA
/end MEASUREMENT
```

6.3.82 MEMORY_LAYOUT

Prototype:

```
/begin MEMORY_LAYOUT    PrgType Address Size Offset
                        { -> IF_DATA}*
/end MEMORY_LAYOUT
```

Parameters:

enum PrgType: Description of the program segments divided into:

PRG_CODE =	program code
PRG_DATA =	program data
PRG_RESERVED =	other

long Address: Initial address of the program segment to be described.

long Size: Length of the program segment to be described.

long[5] Offset: BOSCH feature: In special ECU programs, so-called 'mirrored segments' may occur (see Figure 8). A mirrored segment is a copy of another program segment. During adjustment the data changes are introduced in the relevant memory segment as well as in all mirrored segments.

Optional parameters

-> IF_DATA: Date record to describe the interface specific description data used at ASAP1b device. The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the control unit supplier or the interface module supplier.

Description:

This data record is used to describe an ECU program. The description indicates how the emulation memory is divided into the individual segments.

Example:

See also Figure 8.

```
/begin MEMORY_LAYOUT    PRG_RESERVED
                        0x0000   0x0400   -1       -1       -1    -1    -1
/end MEMORY_LAYOUT
/begin MEMORY_LAYOUT    PRG_CODE
                        0x0400   0x3C00   -1       -1       -1    -1    -1
/end MEMORY_LAYOUT
/begin MEMORY_LAYOUT    PRG_DATA
                        0x4000   0x0200   0x10000  0x20000  -1    -1    -1
/end MEMORY_LAYOUT
```

```
/begin MEMORY_LAYOUT    PRG_DATA
                        0x4200     0x0E00   -1        -1        -1    -1    -1
/end MEMORY_LAYOUT
/begin MEMORY_LAYOUT    PRG_DATA
                        0x14200    0x0E00   -1        -1        -1    -1    -1
/end MEMORY_LAYOUT
/begin MEMORY_LAYOUT    PRG_DATA
                        0x24200    0x0E00   -1        -1        -1    -1    -1
/end MEMORY_LAYOUT
```

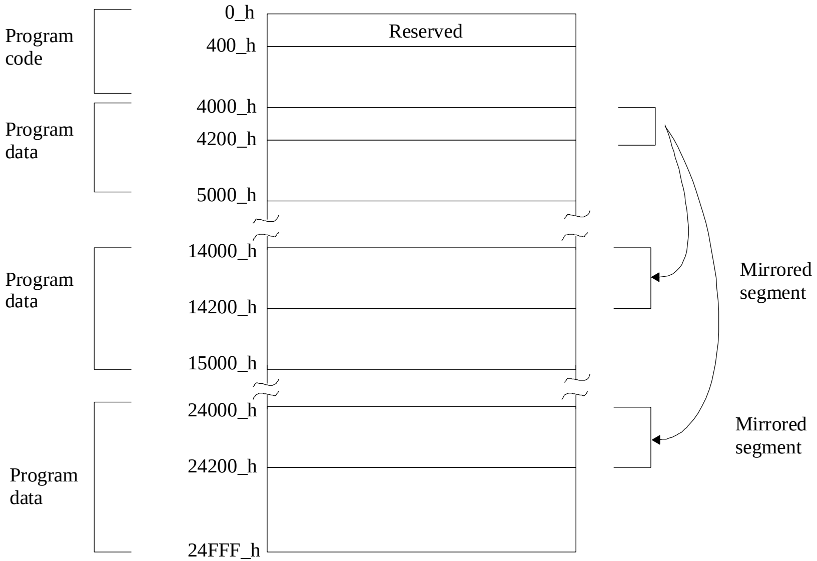


Figure 8: Memory layout (mirrored segments)

6.3.83 MEMORY_SEGMENT

Prototype:

```
/begin  MEMORY_SEGMENT Name LongIdentifier PrgType MemoryType
                        Attribute Address Size Offset
                        { -> IF_DATA}*
/end  MEMORY_SEGMENT
```

Parameters:

ident Name: identifier, reference to IF_DATA Blob is based on this ´name´
string LongIdentifier: comment, description

enum PrgType: PrgTypes:
CODE = program code
DATA = program data allowed for online calbration
OFFLINE_DATA= program data allowed only for offline calibration
VARIABLES = program variables
SERAM = program data for serial emulation
RESERVED = reserved segments

enum MemoryType: Description of the type of memory used
RAM = segment of RAM
EEPROM = segment of EEPROM
EPROM = segment of EPROM
ROM = segment of ROM
REGISTER = segment of CPU registers
FLASH = segment of FLASH

enum Attribute: attributes:
INTERN = internal segment
EXTERN = external segment

long Address: Initial address
long Size: Length of the segment
long[5] Offset: Offset address of mirrored segments

Optional Parameters:

-> IF_DATA: Date record to describe the interface specific description data used at ASAP1b device. The parameters associated with this keyword are described in the ASAP2 metalanguage (in short

A2ML) by the control unit supplier or the interface module supplier

Description:

The new keyword MEMORY_SEGMENT is used to replace the existing keyword MEMORY_LAYOUT. The advantages of MEMORY_SEGMENT are that they are given a name which can be used for references from IF_DATA Blobs and the more accurate description of the memory by memory types and attributes (INTERN and EXTERN).

Used in MOD_PAR. The keywords MEMORY_SEGMENT and MEMORY_LAYOUT can be used in parallel. The parameter Offset is to be used (as within the former MEMORY_LAYOUT) to describe several mirrored segments.

MEMORY_SEGMENTS with the same MemoryType and the same Attribute may not overlap. Also all MEMORY_SEGMENTS with the PrgType CODE, DATA, OFFLINE_DATA, RESERVED may not overlap mutually to get a linear address space for access on calibration data. All other MEMORY_SEGMENTS with different MemoryType or different Attribute may however overlap, e.g. internal and external memory segments.

The following table gives a description for some useful combinations of PrgType and MemoryType and their meanings:

Combination	**Meaning**
CODE / FLASH	Executable code, has to be preserved for download and HEX-file generation,
DATA / FLASH or DATA / EEPROM	Calibration data, can be modified by the user via calibration systems
RESERVED / FLASH	ECU specific code or data, has to be preserved for HEX-file generation but not for download
DATA / RAM	calibration data, will be modified by ECU and calibration systems
OFFLINE_DATA / FLASH	calibration data, will be modified only without ECU access, online calibration is not allowed
VARIABLES / RAM	RAM of the ECU for variables (measurement values and others)
REGISTER / RAM	RAM of the ECU for special purpose values
SERAM / RAM	ECU-RAM section available for serial application. For usage see also: CALIBRATION_METHOD

Note that the MemoryType FLASH has been used as synonym for EPROM and ROM

Example:

```
/begin MEMORY_SEGMENT Data1 “Data internal Flash“
        DATA FLASH INTERN 0x4000 0x0200  0x10000 -1 -1 -1 -1 -1
```

```
/end MEMORY_SEGMENT
/begin MEMORY_SEGMENT Data2 “Data external Flash“
        DATA FLASH EXTERN 0x7000 0x2000 -1 -1 -1 -1 -1
/end MEMORY_SEGMENT
/begin MEMORY_SEGMENT Code1 “Code external Flash“
        CODE FLASH EXTERN 0x9000 0x3000 -1 -1 -1 -1 -1
/end MEMORY_SEGMENT
/begin MEMORY_SEGMENT ext_Ram  “external RAM“
        DATA RAM EXTERN 0x30000 0x1000 -1 -1 -1 -1 -1
/end MEMORY_SEGMENT
/begin MEMORY_SEGMENT int_Ram  “internal RAM“
        DATA RAM INTERN 0x0000 0x0200 -1 -1 -1 -1 -1
/end MEMORY_SEGMENT
/begin MEMORY_SEGMENT Seram1   “emulation RAM 1“
        SERAM RAM EXTERN 0x7000 0x1000 -1 -1 -1 -1 -1
/end MEMORY_SEGMENT
/begin MEMORY_SEGMENT Seram2   “emulation RAM 2“
        SERAM RAM INTERN 0x8000 0x1000 -1 -1 -1 -1 –1
/end MEMORY_SEGMENT
```

6.3.84 MODULE

Prototype:

```
/begin MODULE               Name LongIdentifier
                            [-> A2ML]
                            [-> MOD_PAR]
                            [-> MOD_COMMON]
                            {-> IF_DATA}*
                            {-> CHARACTERISTIC}*
                            {-> AXIS_PTS}*
                            {-> MEASUREMENT}*
                            {-> COMPU_METHOD}*
                            {-> COMPU_TAB}*
                            {-> COMPU_VTAB}*
                            {-> COMPU_VTAB_RANGE}*
                            {-> FUNCTION}*
                            {-> GROUP}*
                            {-> RECORD_LAYOUT}*
                            [-> VARIANT_CODING]
                            [-> FRAME]
                            {-> USER_RIGHTS}*
/end MODULE
```

Parameters:

ident Name:	ASAP device identifier
string LongIdentifier:	comment, description

Optional parameters:

-> A2ML: Format description of the interface-specific parameters.
Attention: The interface-specific parameters must be specified directly after the last mandatory parameter 'long identifier'.

```
/begin MODULE ENGINE_ECU "Comment"
  /begin A2ML
  /end A2ML
   :
/end MODULE
```

-> MOD_PAR: Keyword for the description of module-specific (ASAP device-specific) management data.

-> MOD_COMMON: Module-wide description data

-> IF_DATA: Data record for the description of interface-specific description data (BLOB: binary large object). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

MODULE

-> CHARACTERISTIC: Keyword for the description of the adjustable objects
-> AXIS_PTS: Keyword for the description of the axis points
-> MEASUREMENT: Keyword for the description of the measurement objects
-> COMPU_METHOD: Keyword for the description of the conversion method
-> COMPU_TAB: Keyword for the description of the conversion tables
-> COMPU_VTAB: Keyword for the description of the verbal conversion tables
-> COMPU_VTAB_RANGE: Keyword for the description of range-based verbal conversion tables
-> FUNCTION: Keyword for the description of the functions
-> GROUP: Keyword for the declaration of groups
-> FRAME: Keyword for the declaration of frames
-> RECORD_LAYOUT: Keyowrd for the description of the record layouts
-> VARIANT_CODING: Keyword to describe the variant coding of adjustable objects.
-> USER_RIGHTS : Keyword to reference the groups which constitute access rights.

Description:

The MODULE keyword describes a complete ASAP device with all adjustable and measurement objects, conversion methods and functions. To this, the format description of the interface-specific parameters by the ECU supplier must be added.

Example:

see part C: file engine_ecu.a2l, abs_ecu.a2l

6.3.85 MOD_COMMON

Prototype:

```
/begin MOD_COMMON        Comment
                         [-> S_REC_LAYOUT]
                         [-> DEPOSIT]
                         [-> BYTE_ORDER]
                         [-> DATA_SIZE]
                         [->ALIGNMENT_BYTE]
                         [->ALIGNMENT_WORD]
                         [->ALIGNMENT_LONG]
                         [->ALIGNMENT_FLOAT32_IEEE]
                         [->ALIGNMENT_FLOAT64_IEEE]
/end MOD_COMMON
```

Parameters:

string Comment: comment, description

Optional parameters:

-> S_REC_LAYOUT: Reference to the standard record layout

-> DEPOSIT: Standard deposit mode for axis points: ASBOLUTE or DIFFERENCE

-> BYTE_ORDER: Byte order

-> DATA_SIZE: Data size in bits

-> ALIGNMENT_BYTE: Declares the alignment of bytes in the complete module. The alignment is 1 if parameter is missing.

-> ALIGNMENT_WORD: Declares the alignment of words in the complete module. The alignment is 2 if parameter is missing.

-> ALIGNMENT_LONG: Declares the alignment of longs in the complete module. The alignment is 4 if parameter is missing.

-> ALIGNMENT_FLOAT32_IEEE: Declares the alignment of 32 bit floats in the complete module. The alignment is 4 if parameter is missing.

-> ALIGNMENT_FLOAT64_IEEE: Declares the alignment of 64 bit floats in the complete module. The alignment is 4 if parameter is missing.

Description:

This keyword is used to specify general description data for the module, which are then used as standard in this module. Should other methods be used for an object (e.g. adjustable object or measurement object) of this module, this must then be indicated in the description of the relevant object.

Example:

```
/begin MOD COMMON        "Characteristic maps always deposited in same mode"
        S_REC_LAYOUT     SIEMENS_ABL
        DEPOSIT          ABSOLUTE
        BYTE_ORDER       MSB_LAST
        DATA_SIZE        16
        ALIGNMENT_BYTE   2
/end MOD_COMMON
```

6.3.86 MOD_PAR

Prototype:

```
/begin MOD_PAR          Comment
                        [-> VERSION]
                        [-> ADDR_EPK]
                        [-> EPK]
                        [-> SUPPLIER]
                        [-> CUSTOMER]
                        [-> CUSTOMER_NO]
                        [-> USER]
                        [-> PHONE_NO]
                        [-> ECU]
                        [-> CPU_TYPE]
                        [-> NO_OF_INTERFACES]
                        [-> ECU_CALIBRATION_OFFSET]
                        {-> CALIBRATION_METHOD}*
                        {-> MEMORY_LAYOUT}*
                        {-> MEMORY_SEGMENT}*
                        {-> SYSTEM_CONSTANT}*
/end MOD_PAR
```

Parameters:

string Comment: comment, description relating to the ECU-specific management data

Optional parameters:

-> CALIBRATION_METHOD: Declares the implemented calibration methods in the control unit.
-> MEMORY_SEGMENT: Declares the available memory segments.
-> VERSION: Version identifier
-> ADDR_EPK: Address of EPROM identifier
-> EPK: EPROM identifier
-> SUPPLIER: Manufacturer or supplier
-> CUSTOMER: Firm or customer
-> CUSTOMER_NO: Customer number
-> USER: User
-> PHONE_NO: Phone number of the applications engineer responsible
-> ECU: Control unit
-> CPU_TYPE: CPU
-> ECU_CALIBRATION_OFFSET: Offset that has to be added to each address of a characteristic.
-> NO_OF_INTERFACES: Number of interfaces
-> MEMORY_LAYOUT: Memory layout
-> SYSTEM_CONSTANT: System-defined constants

MOD_PAR

Description:

The MOD_PAR keyword describes the management data to be specified for an ASAP device. Except for the comment all parameters are optional.

MOD_PAR

Example:

```
/begin MOD_PAR                "Please note: provisional release for test purposes only!"
        VERSION               "Test version of 01.02.1994"
        ADDR_EPK              0x45678
        EPK                   EPROM identifier test
        SUPPLIER              "Mustermann"
        CUSTOMER              "LANZ-Landmaschinen"
        CUSTOMER_NO           "0123456789"
        USER                  "A.N.Wender"
        PHONE_NO              "09951 56456"
        ECU                   "Engine control"
        CPU_TYPE              "Motorola 0815"
        NO_OF_INTERFACES 2
        /begin MEMORY_SEGMENT ext_Ram  “external RAM“
                DATA RAM EXTERN 0x30000 0x1000 -1 -1 -1 -1 -1
                /begin IF_DATA ASAP1B_KWP2000
                /* ADDRESS_MAPPING     orig_addr     mapping_addr length */
                  ADDRESS_MAPPING      0x4000        0x6000              0x0200
                /end IF_DATA
        /end MEMORY_SEGMENT
        /begin MEMORY_LAYOUT   PRG_RESERVED
                              0x0000 0x0400 -1 -1 -1 -1 -1
        /end MEMORY_LAYOUT
        /begin MEMORY_LAYOUT   PRG_CODE
                              0x0400 0x3C00 -1 -1 -1 -1 -1
        /end MEMORY_LAYOUT
        /begin MEMORY_LAYOUT   PRG_DATA
                              0x4000 0x5800 -1 -1 -1 -1 -1
        /end MEMORY_LAYOUT
        SYSTEM_CONSTANT "CONTROLLERx constant1" "0.33"
        SYSTEM_CONSTANT "CONTROLLERx constant2" "2.79"
/end MOD_PAR
```

6.3.87 MONOTONY

Prototype:

MONOTONY Monotony

Parameters:

enum Monotony: Description of the monotony:

MON_INCREASE:	monotonously increasing
MON_DECREASE:	monotonously decreasing
STRICT_INCREASE:	strict monotonously increasing
STRICT_DECREASE:	strict monotonously decreasing

Description:

This keyword can be used to specify the monotony of an adjustment object. The monotony is always related to an axis (see keyword "AXIS_DESCR"). With each adjustment operation the application system (user interface) verifies whether the monotony is guaranteed. Changes that do not correspond to the monotony are not allowed.

Example:

```
MONOTONY                MON_INCREASE
```

6.3.88 MULTIPLEX

Prototype:

MULTIPLEX Startbit DataSize Tag

Parameters:

int Startbit:	The 'mode-signal' is included in CAN-message at offset 'Startbit' (in bits, index of first bit of message data is 0).
int DataSize:	Size of 'mode-signal'.
long Tag:	The corresponding measurement data are valid only if the value of 'mode'signal' included in CAN-message agrees with the value specified with 'Tag'.

Description:

In 'Standard-Mode' a fixed data segment is assigned to each data object of CAN-message. 'Multiplexing' is a special method to transmit different data objects using the same CAN-message data segment. For that purpose an additional 'mode-signal' is used indicating the data object contained in multiplexed data segment.

Example:

```
/begin IF_DATA   ASAP1B_CAN
        /begin KP_BLOB          „MOTOINFO“  0x123  8  „MOTOR-SG“  5  8
                                MULTIPLEX  48  16  0x4321
        /end KP_BLOB
/end IF_DATA
```

This example describes the measurement object „MOTO-INFO“ contained in data segment bit5...bit12 of CAN-message transmitted from „MOTOR-SG“. This data segment is a multiplexed segment. That means, that the measurement object „MOTO-INFO“ is contained in data segment bit5...bit12 only if the 'mode-signal' (data segment bit48...bit63) agrees to the value 0x4321.

6.3.89 NO_AXIS_PTS_X/_Y /_Z

Prototype:

NO_AXIS_PTS_X/_Y/_Z	Position Datatype

Parameters:

int Position:	Position of the number of axis points in the deposit structure
datatype Datatype:	Data type of the number of axis points

Description:

Description of the number of axis points in an adjustable object

Example:

```
NO_AXIS_PTS_X                2 UWORD
```

6.3.90 NO_OF_INTERFACES

Prototype:

NO_OF_INTERFACES	Num

Parameters:

int Num:	Number of interfaces

Description:

Keyword for the number of interfaces

Example:

```
NO_OF_INTERFACES 2
```

6.3.91 NO_RESCALE_X/_Y/_Z

Prototype:

NO_RESCALE_X/_Y/_Z	Position Datatype

Parameters:

int Position:	position of the actual number of rescale axis point value pairs in the deposit structure (description of sequence of elements in the data record).
datatype DataType:	Data type of the number of rescale axis point value pairs

Description:

Actual number of rescaling axis point value pairs. Used in RECORD_LAYOUT.

Example:

NO_RESCALE_X	1 UBYTE

6.3.92 NUMBER

Prototype:

NUMBER	Number

Parameters:

int Number:	Number of values (array of values) or characters (string).

Description:

In the CHARACTERISTIC data record, this keyword can be used to specify the number of values and characters for the adjustable object types 'array of values' (VAL_BLK) and 'string' (ASCII) respectively.

Example:

```
NUMBER                    7
```

6.3.93 OFFSET_X/_Y/_Z

Prototype:

OFFSET_X/_Y/_Z Position Datatype

Parameters:

int Position: Position of the 'offset' parameter in the deposit structure to compute the X-axis points for fixed characteristic curves and fixed characteristic maps.

datatype Datatype: Data type of the 'offset' parameter.

Description:

Description of the 'offset' parameter in the deposit structure to compute the axis points for fixed characteristic curves and fixed characteristic maps (see also keyword FIX_AXIS_PAR). The axis points for fixed characteristic curves or fixed characteristic maps are derived from the two 'offset' and 'shift' parameters as follows:

$X_i = \text{Offset} + (i - 1)*2^{\text{Shift}}$ $i = \{ 1...\text{numberofaxispts} \}$

or

$Y_k = \text{Offset} + (k - 1)*2^{\text{Shift}}$ $k = \{ 1...\text{numberofaxispts} \}$

or

$Z_m = \text{Offset} + (m - 1)*2^{\text{Shift}}$ $m = \{ 1...\text{numberofaxispts} \}$

Example:

OFFSET_X 16 UWORD

6.3.94 OUT_MEASUREMENT

Prototype:

```
/begin OUT_MEASUREMENT { Identifier } *
/end OUT_MEASUREMENT
```

Parameters:

ident Identifier:	Identifier of output quantity of respective function (reference to measurement object).

Description:

This keyword can be used to define output quantities of respective function.
Note: OUT_MEASUREMENT may only refer to objects of type MEASUREMENT.

Example:

```
/begin OUT_MEASUREMENT                    OK_FLAG  SENSOR_FLAG
/end OUT_MEASUREMENT
```

6.3.95 PHONE_NO

Prototype:

PHONE_NO	Telnum

Parameters:

string Telnum:	phone number

Description:

This keyword is used to specify a phone number, e.g. of the applications engineer responsible.

Example:

```
PHONE_NO                    "09498 594562"
```

6.3.96 PROJECT

Prototype:

```
/begin PROJECT              Name LongIdentifier
                            [-> HEADER]
                            {-> MODULE}*
/end PROJECT
```

Parameters:

ident Name:	Project identifier in the program
string LongIdentifier:	Comment, description

Optional parameters:

-> HEADER:	Project header
-> MODULE:	This keyword is used to describe the module (ASAP device) belonging to the project.

Description:

Project description with project header and all ASAP devices belonging to the project. The PROJECT keyword covers the description of several control units, and possibly also of several suppliers.

Example:

```
/begin PROJECT              RAPE-SEED ENGINE
                            "Engine tuning for operation with rape oil"
        /begin HEADER       "see also specification XYZ of 01.02.1994"
                            VERSION             "BG5.0815"
                            PROJECT_NO          M4711Z1
        /end HEADER

        /include ENGINE_ECU.A2L                 /* Include for engine control module */
        /include ABS_ECU.A2L                    /* Include for ABS module */
/end PROJECT
```

6.3.97 PROJECT_NO

Prototype:

PROJECT_NO	ProjectNumber

Parameters:

ident ProjectNumber:	Short identifier of the project number

Description:

String used to identify the project number with maximum MAX_IDENT (at present MAX_IDENT = 10) characters.

Example:

```
PROJECT_NO                    M4711Z1
```

6.3.98 RASTER

Prototype:

```
/begin RASTER           RasterName  ShortName  RasterID  ScalingUnit  Rate
/end  RASTER
```

Parameters:

string RasterName: Event channel name (name for one sample rate of an ECU supported data acquisition mechanism, i.e. the ASAP1a-CCP)

string ShortName: Short display name of the event channel name. Recommendation : The string length shall not exceed 9 characters

int RasterID: Event channel No., e.g. for ASAP1a-CCP START_STOP. This parameter can be used as a reference, therefore the number must be unique among all declared RASTER.

int ScalingUnit: Period definition : basic scaling unit (see Table 4 : Codes for scaling unit)

long Rate: ECU sample rate of the event channel, period definition based on the scaling unit

Description:

Applied for ECU supported data acquisition mechanisms, i.e. the DAQ lists of the ASAP1a-CCP (CAN Calibration Protocol) or the monitoring copy routines for the microcontroller internal RAM memory when using a memory probe. Such mechanisms typically provide „services“ based on a sample rate (periodic event) or a cyclic event.
Each available „service“ can be described by one RASTER (data acquisition event supported by the ECU) within IF_DATA(Module) and then can be referenced within the ASAP1b-QP_BLOB statement of the SOURCE keyword.

Note : This keyword is only used in the context of the IF_DATA(MODULE) applied for the ASAP1a-CCP.

Example:

```
/begin  RASTER          "Segment synchronous cylinder" "CYL1"  1  103  1
/end  RASTER
```

6.3.99 READ_ONLY

Prototype:

READ_ONLY

Description:

This keyword is used to indicate that an adjustable object cannot be changed (but can only be read).

Example:

```
/begin CHARACTERISTIC      KI "I-share for speed limitation"
                           VALUE         /* type: fixed value */
                           0x408F        /* address */
                           DAMOS_FW      /* deposit */
                           0.0           /* max_diff */
                           FACTOR01      /* conversion */
                           0.0           /* lower limit */
                           255.0         /* upper limit */

     /* interface-specific parameters: address location, addressing */
     /begin IF_DATA "DIM"  EXTERNAL DIRECT /end IF_DATA

     /begin FUNCTION_LIST  V_LIM                  /*Reference to functions */
     /end FUNCTION_LIST
     READ_ONLY
/end CHARACTERISTIC
```

6.3.100 READ_WRITE

Prototype:

READ_WRITE

Description:

This keyword is used to mark a measurement object to be writeable.

Example:

```
/begin MEASUREMENT      N                   /* name */
                        "Engine speed"      /* long identifier */
                        UWORD               /* datatype */
                        R_SPEED_3           /* conversion */
                        2                   /* resolution */
                        2.5                 /* accuracy */
                        120.0               /* lower limit */
                        8400.0              /* upper limit */
        READ_WRITE
        /begin IF_DATA ISO      SND 0x10 0x00 0x05 0x08 RCV 4  long /end IF_DATA
/end MEASUREMENT
```

6.3.101 RECORD_LAYOUT

Prototype:

```
/begin RECORD_LAYOUT      Name
                          [-> FNC_VALUES]
                          [-> IDENTIFICATION]
                          [-> AXIS_PTS_X]
                          [-> AXIS_PTS_Y]
                          [-> AXIS_PTS_Z]
                          [-> AXIS_RESCALE_X]
                          [-> AXIS_RESCALE_Y]
                          [-> AXIS_RESCALE_Z]
                          [-> NO_AXIS_PTS_X]
                          [-> NO_AXIS_PTS_Y]
                          [-> NO_AXIS_PTS_Z]
                          [-> NO_RESCALE_X]
                          [-> NO_RESCALE_Y]
                          [-> NO_RESCALE_Z]
                          [-> FIX_NO_AXIS_PTS_X]
                          [-> FIX_NO_AXIS_PTS_Y]
                          [-> FIX_NO_AXIS_PTS_Y]
                          [-> SRC_ADDR_X]
                          [-> SRC_ADDR_Y]
                          [-> SRC_ADDR_Z]
                          [-> RIP_ADDR_X]
                          [-> RIP_ADDR_Y]
                          [-> RIP_ADDR_Z]
                          [-> RIP_ADDR_W]
                          [-> SHIFT_OP_X]
                          [-> SHIFT_OP_Y]
                          [-> SHIFT_OP_Z]
                          [-> OFFSET_X]
                          [-> OFFSET_Y]
                          [-> OFFSET_Z]
                          [-> DIST_OP_X]
                          [-> DIST_OP_Y]
                          [-> DIST_OP_Z]
                          [->ALIGNMENT_BYTE]
                          [->ALIGNMENT_WORD]
                          [->ALIGNMENT_LONG]
                          [->ALIGNMENT_FLOAT32_IEEE]
                          [->ALIGNMENT_FLOAT64_IEEE]
                          {-> RESERVED}*
/end RECORD_LAYOUT
```

Parameters:

ident Name: Identification of the record layout, which is referenced via this 'name'.

Optional parameters:

-> FNC_VALUES: This keyword describes how the table values (function values) of the adjustable object are deposited in memory.

-> IDENTIFICATION: This keyword is used to describe that an 'identifier' (see BOSCH: C-DAMOS) is deposited in a specific position in the adjustable object.

only characteristic curves or characteristic maps:

-> AXIS_PTS_X: This keyword describes where the X-points are deposited in memory.

-> AXIS_PTS_Y: This keyword describes where the Y-points are deposited in memory.

-> AXIS_PTS_Z: This keyword describes where the Z-points are deposited in memory.

-> NO_AXIS_PTS_X: This keyword describes in which position the parameter 'number of X-axis points' is deposited in memory.

-> NO_AXIS_PTS_Y: This keyword describes in which position the parameter 'number of Y-axis points' is deposited in memory.

-> NO_AXIS_PTS_Z: This keyword describes in which position the parameter 'number of Z-axis points' is deposited in memory.

-> FIX_NO_AXIS_PTS_X: This keyword indicates that all characteristic curves or characteristic maps relating to the X-axis points are allocated a fixed number of axis points. In a RECORD_LAYOUT data record, this keyword may not be used simultaneously with the keyword 'NO_AXIS_PTS_X(!!).

-> FIX_NO_AXIS_PTS_Y: This keyword indicates that all characteristic curves or characteristic maps relating to the Y-axis points are allocated a fixed number of axis points. In a RECORD_LAYOUT data record, this keyword may not be used simultaneously with the keyword 'NO_AXIS_PTS_Y (!!).

-> FIX_NO_AXIS_PTS_Z: This keyword indicates that all characteristic curves or characteristic maps relating to the Z-axis points are allocated a fixed number of axis points. In a RECORD_LAYOUT data record, this keyword may not be used simultaneously with the keyword 'NO_AXIS_PTS_Z (!!).

-> SRC_ADDR_X: This keyword describes in which position the address of the input quantity of the X-axis points is deposited in memory.

-> SRC_ADDR_Y: This keyword describes in which position the address of the input quantity of the Y-axis points is deposited in memory.

-> SRC_ADDR_Z: This keyword describes in which position the address of the input quantity of the Z-axis points is deposited in memory.

-> RIP_ADDR_X: Future record layouts: When the ECU program accesses a characteristic curve it determines an output value based on an input quantity. First it searches the adjacent axis points of the current value of the input quantities (X_i, X_{i+1}). The output

value is derived from these axis points and the allocated table values by means of interpolation. This produces an 'intermediate result' known as the RIP_X quantity (Result of Interpolation), which describes the relative distance between the current value and the adjacent axis points:
RIP_X = (X(t) - X_i)/(X_{i+1} - X_i).

This keyword is used to describe in which position the address of this RIP_X quantity is deposited, which contains the current value of the ECU-internal interpolation.

-> RIP_ADDR_Y/_Z: See RIP_ADDR_Y, but for Y/Z-axis points.

-> RIP_ADDR_W: Final result (table value) of the ECU-internal inter- polation.

only for fixed curves or fixed maps (at the request of Mr Hünerfeld):

-> SHIFT_OP_X: Shift operand to compute the X-axis points

-> SHIFT_OP_Y: Shift operand to compute the Y-axis points

-> SHIFT_OP_Z: Shift operand to compute the Z-axis points

-> OFFSET_X: Offset to compute the X-axis points

-> OFFSET_Y: Offset to compute the Y-axis points

-> OFFSET_Z: Offset to compute the Z-axis points

-> DIST_OP_X: 'Distance' parameter to compute the X-axis points

-> DIST_OP_Y: 'Distance' parameter to compute the Y-axis points

-> DIST_OP_Z: 'Distance' parameter to compute the Z-axis points

-> RESERVED: This keyword can be used to skip specific elements in the adjustable object whose meaning must not be interpreted by the application system (e.g. for extensions: new parameters in the adjustable objects).

-> AXIS_RESCALE_X: This keyword describes where the rescale mapping for the x-axis is deposited in memory.

-> AXIS_RESCALE_Y: This keyword describes where the rescale mapping for the y-axis is deposited in memory.

-> AXIS_RESCALE_Z: This keyword describes where the rescale mapping for the z-axis is deposited in memory.

-> NO_RESCALE_X: This keyword describes at which position the parameter ‘actual number of rescale pairs’ for the x-axis is deposited (see AXIS_RESCALE).

-> NO_RESCALE_Y: This keyword describes at which position the parameter ‘actual number of rescale pairs’ for the y-axis is deposited (see AXIS_RESCALE).

-> NO_RESCALE_Z: This keyword describes at which position the parameter ‘actual number of rescale pairs’ for the z-axis is deposited (see AXIS_RESCALE).

-> ALIGNMENT_BYTE: Declares the alignment of bytes for all characteristics who use this record layout.. The alignment is 1 if parameter is missing.

-> ALIGNMENT_WORD: Declares the alignment of words for all characteristics who use this record layout. The alignment is 2 if parameter is missing.

-> ALIGNMENT_LONG: Declares the alignment of longs for all characteristics who use this record layout. The alignment is 4 if parameter is missing.

-> ALIGNMENT_FLOAT32_IEEE: Declares the alignment of 32 bit floats for all characteristics who use this record layout. The alignment is 4 if parameter is missing.

-> ALIGNMENT_FLOAT64_IEEE: Declares the alignment of 64 bit floats for all characteristics who use this record layout. The alignment is 4 if parameter is missing.

Description:

The 'RECORD_LAYOUT' keyword is used to specify the various *record layouts* of the adjustable objects in the memory. The structural buildup of the various adjustable object types must be described in such a way that a standard application system will be able to process all adjustable object types (reading, writing, operating point display etc.).

Important:

To describe the record layouts, use is made of a predefined list of parameters which may be part of an adjustable object (characteristic) in the emulation memory. This list represents the current status of the record layouts. With each change or extension of the record layouts contained in this predefined list of parameters the ASAP2 description file format must be modified accordingly.

Example:

```
/begin RECORD_LAYOUT      DAMOS_KF
        FNC_VALUES        7 SWORD COLUMN_DIR DIRECT
        AXIS_PTS_X        3 SWORD INDEX_INCR DIRECT
        AXIS_PTS_Y        6 UBYTE INDEX_INCR DIRECT
        NO_AXIS_X         2 UBYTE
        NO_AXIS_Y         5 UBYTE
        SRC_ADDR_X        1
        SRC_ADDR_Y        4
       ALIGNMENT_BYTE     2
/end RECORD_LAYOUT

/begin RECORD_LAYOUT         RESCALE_SST
       NO_RESCALE_X          1 UBYTE
       RESERVED              2 UBYTE
       AXIS_RESCALE_X        3 UBYTE 5 INDEX_INCR DIRECT
/end RECORD_LAYOUT
```

6.3.102 REF_CHARACTERISTIC

Prototype:

```
/begin REF_CHARACTERISTIC                    { Identifier } *
/end REF_CHARACTERISTIC
```

Parameters:

ident Identifier: Identifier of those adjustable objects that are refered to respective function or group.

Description:

This keyword can be used to define some adjustable objects that are referenced in respective function or group.
Note: REF_CHARACTERISTIC may only refer to objects of type CHARACTERISTIC or AXIS_PTS.

Example:

```
/begin REF_CHARACTERISTIC        ENG_SPEED_CORR_CURVE
/end REF_CHARACTERISTIC
```

6.3.103 REF_GROUP

Prototype:

```
/begin REF_GROUP                                    { Identifier } *
/end REF_GROUP
```

Parameters:

ident Identifier: Identifier of those groups which are refered in USER_RIGHTS

Description:

This keyword can be used to refer groups which control the access rights of users logging into an .MCD system.

Example:

```
/begin REF_GROUP                  GROUP_1   GROUP_2
/end REF_GROUP
```

6.3.104 REF_MEASUREMENT

Prototype:

/begin REF_MEASUREMENT { Identifier } *
/end REF_MEASUREMENT

Parameters:

ident Identifier: Identifier of those measurement quantities which are refered to the group.

Description:

This keyword can be used to define measurement quantities which are member of the respective function.

Example:

/begin REF_MEASUREMENT LOOP_COUNTER TEMPORARY_1
/end REF_MEASUREMENT

6.3.105 REF_MEMORY_SEGMENT

Prototype:

REF_MEMORY_SEGMENT Name

Parameters:

ident Name:	Name of memory segments

Description:

The reference to a memory segment is needed in characteristics and measurements. The memory segment, the characteristic belongs to can not be detected by the address itself in the case of overlapping memory segments.
Used in CHARACTERISTIC, AXIS_PTS, MEASUREMENT.

Example:

REF_MEMORY_SEGMENT Data1

6.3.106 RESERVED

Prototype:

RESERVED	Position Datatype

Parameters:

int Position:	Position of the reserved parameter in the deposit structure
datasize DataType:	Word length of the reserved parameter.

Description:

This keyword can be used to skip specific elements in an adjustable object whose meaning must not be interpreted by the application system (e.g. for extensions: new parameters in the adjustable objects).

Example:

```
RESERVED                7 LONG
```

6.3.107 RIGHT_SHIFT

Prototype:

RIGHT_SHIFT	Bitcount

Parameters:

long bitcount:	Shift ‚bitcount‘ bits to the right

Description:

The RIGHT_SHIFT keyword is only used within the BIT_OPERATION keyword. See description of BIT_OPERATION.

6.3.108 RIP_ADDR_X/_Y/_Z/_W

Prototype:

RIP_ADDR_X/_Y/_Z/_W Position Datatype

Parameters:

int Position:	Position of the address to the result of the ECU-internal interpolation (see below) in the deposit structure.
datatype Datatype:	Data type of the address. Remark: Relating to version 1.0 of ASAP2-Specification this is an additional parameter.

Description:

The description of this parameter should be based on the example of a characteristic curve (RIP: Result of Interpolation).
When the ECU program accesses the characteristic curve it first determines the adjacent axis points of the current value of the input quantity (see Figure 9: X_i, X_{i+1}). The output value is derived from these axis points and the two allocated table values by means of interpolation. This produces as intermediate results the quantities RIP_X and RIP_Y, which describe the distance between the current value and the adjacent axis points:

$$RIP\_X = (X_{current} - X_i)/(X_{i+1} - X_i)$$

For a characteristic map the ECU program determines this intermediate result both in the X-direction and in the Y-direction. For a characteristic cuboid the result in the direction of all three axes are calculated.

$$RIP\_Y = (Y_{current} - Y_k)/(Y_{k+1} - Y_k)$$

$$RIP\_Z = (Z_{current} - Z_m)/(Z_{m+1} - Z_m)$$

For a characteristic curve the result of the interpolation is calculated as follows:

$$RIP\_W = W_i + (RIP\_X * (W_{i+1} - W_i)$$

for a characteristic map as follows:

$$RIP\_W = (W_{i,k} * (1 - RIP\_X) + W_{i+1,k} * RIP\_X)) * (1 - RIP\_Y) +$$

$$(W_{i,k+1} * (1 - RIP\_X) + W_{i+1,k+1} * RIP\_X)) * RIP\_Y$$

and for a characteristic cuboid as follows:

Interpolation for the map Z = m

$RIP\_W_{m} = (W_{i,k,m} * (1 - RIP\_X) + W_{i+1,k,m} * RIP\_X)) * (1 - RIP\_Y) +$

$(W_{i,k+1,m} * (1 - RIP\_X) + W_{i+1,k+1,m} * RIP\_X)) * RIP\_Y$

Interpolation for the map Z = m+1

$RIP\_W_{m+1} = (W_{i,k,m+1} * (1 - RIP\_X) + W_{i+1,k,m+1} * RIP\_X)) * (1 - RIP\_Y) +$

$(W_{i,k+1,m+1} * (1 - RIP\_X) + W_{i+1,k+1,m+1} * RIP\_X)) * RIP\_Y$

Interpolation in Z direction between the two points RIP_W_{m} and RIP_W_{m+1}.

$RIP\_W = RIP\_W_{m} +(RIP\_Z*(RIP\_W_{m+1} - RIP\_W_{m})$

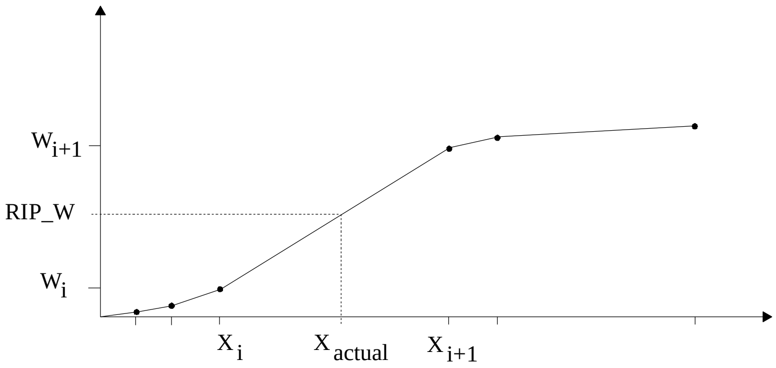


Figure 9: Linear interpolation for a characteristic curve

Example:

RIP_ADDR_X 19 UWORD

6.3.109 ROOT

Prototype:

ROOT

Parameters:

none

Description:

This keyword ROOT indicates that the related group is presented as a root of a navigation tree in the group selection mechanism of the MCD system. The keyword ROOT can indicate that groups refered to this root group constitute a grouping mechanism.

Example:

```
/begin GROUP                    SOFTWARE_COMPONENTS
      "assignment of the definitions to C files"
      ROOT
      /begin SUB_GROUP   INJE   C6TD
      /end SUB_GROUP
/end GROUP
```

6.3.110 SEED_KEY

Prototype:

```
/begin SEED_KEY            CalDll  DaqDll  PgmDll
/end SEED_KEY
```

Parameters:

string CalDll: Reference to DLL file name for CAL (ECU Calibration) access priviledge.

string DaqDll: Reference to DLL file name for DAQ (ECU Data Acquisition) access priviledge.

string PgmDll: Reference to DLL file name for PGM (ECU Flash Programming) access priviledge.

Description:

Description of the authentification process (standardized interface to the confidential seed-key algorithms for serial ECU access). References to dynamic link libraries (DLL) with a standard application interface (API), contain the authentification algorithms. The ECU supplier provides the DLLs, the MCD systems reads them to enable calibration, data acquisition or reprogramming access to the ECU.

Note : This keyword is only used in the context of the IF_DATA(MODULE) applied for the ASAP1a-CCP.

Note : The usage of the DLL is based on a standard API definition published in chapter 8.7.

Example:

```
/begin  SEED_KEY           "access.dll"   "access.dll"    "access.dll"
/end SEED_KEY
```

6.3.111 SIGN_EXTEND

Prototype:

SIGN_EXTEND

Parameters:

Description:

The SIGN_EXTEND keyword is only used within the BIT_OPERATION keyword. See description of BIT_OPERATION.

6.3.112 SOURCE

Prototype:

```
/begin SOURCE                 Name ScalingUnit Rate
                              [-> DISPLAY_IDENTIFIER]
                              [-> QP_BLOB QP_Data]
/end SOURCE
```

Parameters:

ident Name: Identifier of measurement source (data acquisition mode).

int ScalingUnit: This parameter defines the basic scaling unit. The following parameter 'Rate' relates on this scaling unit. The value of ScalingUnit is coded as shown in ‘Table 4: Codes for scaling units (CSE)’ (page 130).

long Rate: The maximum refresh rate of the concerning measurement source in the control unit. The unit is defined with parameter 'ScalingUnit'.

Optional parameters:

-> DISPLAY_IDENTIFIER: Display name for the data acquisition channel of the SOURCE.

-> QP_BLOB: Interface-specific description data (BLOB: binary large object) used at ASAP1b device at call of the command InitRead(). The parameters associated with this keyword are described in the ASAP2 metalanguage (in short A2ML) by the ECU supplier or the interface module supplier.

Description:

This keyword can be used to describe different 'sources' of measurement, e.g. different data acquisition modes (e.g. time synchronous 10 msec, time synchronous 50 msec etc.).

Example:

```
/* refresh rate of acquisition mode 'Time35' is 35 milli-seconds, */
/* (value of ScalingUnit = 3 corrensponds to 1 msec) */
/begin SOURCE                 Time35   3   35
/end SOURCE
```

6.3.113 SHIFT_OP_X/_Y/_Z

Prototype:

SHIFT_OP_X/_Y/_Z Position Datatype

Parameters:

int Position:	Position of the shift operand in the deposit structure.
datatype Datatype:	Data type of the shift operand.

Description:

Description of the shift operand in the deposit structure to compute the axis points for fixed characteristic curves and fixed characteristic maps (see also keyword FIX_AXIS_PAR). The axis points distribution for fixed characteristic curves or fixed characteristic maps is derived from the two 'offset' and 'shift' parameters as follows:

$X_i = \text{Offset} + (i - 1)*2^{\text{Shift}}$ $i = \{ 1...\text{numberofaxispts} \}$

or

$Y_k = \text{Offset} + (k - 1)*2^{\text{Shift}}$ $k = \{ 1...\text{numberofaxispts} \}$

or

$Z_m = \text{Offset} + (m - 1)*2^{\text{Shift}}$ $m = \{ 1...\text{numberofaxispts} \}$

Example:

SHIFT_OP_X 21 UWORD

6.3.114 SRC_ADDR_X/_Y/_Z

Prototype:

SRC_ADDR_X/_Y/_Z	Position Datatype

Parameters:

int Position:	Position of the address of the input quantity in the deposit structure.
datatype Datatype:	Data type of the address. Remark: Relating to version 1.0 of ASAP2-Specification this is an additional parameter. The appropriate parameter in the corresponding measuring channel data record is not any longer used.

Description:

Description of the address of the input quantity in an adjustable object

Example:

SRC_ADDR_X	1 UWORD

6.3.115 SUB_FUNCTION

Prototype:

/begin SUB_FUNCTION	{ Identifier } *
/end SUB_FUNCTION	

Parameters:

ident Identifier:	Reference to function record. This function record is declared as subfunction of resprectiv function.

Description:

This keyword can be used to define the hierarchical structur of functions.
Note: SUB_FUNCTION may only refer to objects of type FUNCTION.

Example:

/begin SUB_FUNCTION	ID_ADJUSTM_SUB
/end SUB_FUNCTION	

6.3.116 SUB_GROUP

Prototype:

```
/begin SUB_GROUP              { Identifier } *
/end SUB_GROUP
```

Parameters:

ident Identifier:	Reference to a group record. This group record is declared as sub-group of the respective GROUP.

Description:

This keyword can be used to define the hierarchical structure of groups. In particular, a set of groups referenced from a root group (with optional keyword ROOT) constitute a grouping mechanism.

Example:

```
/begin  SUB_GROUP             ID_ADJUSTM_SUB
/end SUB_GROUP
```

6.3.117 SUPPLIER

Prototype:

SUPPLIER	Manufacturer

Parameters:

string Manufacturer:	Name of the ECU manufacturer

Description:

String used to identify the manufacturer or supplier.

Example:

```
SUPPLIER                "Smooth and Easy"
```

6.3.118 SYSTEM_CONSTANT

Prototype:

SYSTEM_CONSTANT Name Value

Parameters:

string Name:	system constant identifier
string Value:	value of the system constant as a string

Description:

System-defined constant.

Example:

```
SYSTEM_CONSTANT          "CONTROLLER_CONSTANT12" "2.7134"
```

6.3.119 S_REC_LAYOUT

Prototype:

S_REC_LAYOUT	Name

Parameters:

ident Name:	Name of the standard record layout (see RECORD_LAYOUT)

Description:

This keyword can be used to specify the name of a standard record layout which will then apply to all characteristics in the entire module. Exceptions can be specified for the relevant characteristics.
Note: S_REC_LAYOUT may only refer to objects of type RECORD_LAYOUT.

Example:

```
S_REC_LAYOUT            SIEMENS_ABL         /* Siemens record layout */
```

6.3.120 USER

Prototype:

USER UserName

Parameters:

string UserName: Name of the user

Description:

Specification of the user name.

Example:

USER "Nigel Hurst"

6.3.121 USER_RIGHTS

Prototype:

```
/begin USER_RIGHTS          UserLevelId
                            {-> REF_GROUP} *
                            [-> READ_ONLY]
/end USER_RIGHTS
```

Parameters:

ident UserLevelId: When a user logs into the MCD system, a UserLevelId is assigned.

Optional parameters:

-> REF_GROUP: Reference to groups.
Only the CHARACTERSITIC and MEASUREMENT members of the referenced groups including the members of nested subgroups (and functions nested in such groups) are visible to the user of the MCD system. If the READ_ONLY attribute is set, the CHARACTERISTICs are visible but not available for calibration (not tuneable).
The restrictions are applied by the MCD system as a global filter in the user interface (active for all manual selection or calibration operations). When navigating by GROUPs, only the GROUPs declared in USER_RIGHTS need to be provided in the selection list.

-> READ_ONLY: This keyword can be used to define all characteristics of the groups referenced by this USER_RIGHT statement as READ_ONLY (not tunable).

Use Case : A group can be defined to specify a set of characteristics to be not tunable for a group of users (control of access rights). In order to achieve this, the group is referenced in a USER_RIGHT statement with the READ_ONLY attribute, related to the user group.
When a login to the MCD system identifies the user as member of a group for which the USER_RIGHT statement contains the READ_ONLY attribute, all CHARACTERISTICs of this group shall be treated as if the READ_ONLY attribute was directly related to the CHARACTERISTICs.

Description:

This keyword can be used to define groups accessible for certain users. All USER-RIGHTS groups are listed to the user who can select one of these groups. All measurements and characteristics belonging to that group and its subgroubs (and sub-subgroups and so on) are accessible (i.e. visible) to the user. The keyword READ_ONLY is used to define the refered group(s) as containing characteristics that are only readable but not writable (i.e. they can not be adjusted). This property is also inherited by subgroups, i.e. if a group is marked as READ_ONLY all its subgroups (with respect to that USER RIGHT) are also only READ_ONLY.

Example:

```
/begin  USER_RIGHTS          application_engineers
        /begin REF_GROUP   group_1 /end REF_GROUP
/end  USER_RIGHTS

/begin  USER_RIGHTS          measurement_engineers
        /begin REF_GROUP   group_1  /end REF_GROUP
        READ_ONLY
/end USER_RIGHTS

/begin  GROUP                group_1
        /begin REF_CHARACTERISTIC
                             KF1  KF2
        /end REF_CHARACTERISTIC
        /begin REF_MEASUREMENT
                             NMOT TMOT
        /end REF_MEASUREMENT
/end  GROUP
```

6.3.122 VARIANT_CODING

Prototype:

```
/begin VARIANT_CODING      [->VAR_SEPARATOR]
                           [->VAR_NAMING]
                           {->VAR_CRITERION } *
                           {->VAR_FORBIDDEN_COMB } *
                           {->VAR_CHARACTERISTIC } *
/end VARIANT_CODING
```

Optional Parameters:

->VAR_SEPARATOR: This keyword can be used to define the separating symbol between the two parts of adjustable objects name: 1.) identifier 2.) variant extension.
Remark: The identifier of description record of variant coded adjustable objects contains no variant extension. This extension is needed to distinguish the variants at MCD.

->VAR_NAMING: This keyword defines the format of variant externsion (index) of adjustable objects name (index is used at MCD to distinguish the variants).

->VAR_CRITERION: This keyword describes a variant criterion, i.e. some adjustable objects are multiple deposited in control unit software corresponding to the enumeration of variant criterion values.

->VAR_FORBIDDEN_COMB: This keyword describes a forbidden combination of different variant criteria.

->VAR_CHARACTERISTIC: This keyword defines one adjustable object to be variant coded, i.e. this adjustable objects is multiple deposited in control unit software corresponding to the assigned variant criteria.

Description:

The information of variant coding is grouped to this keyword. Variant coding means, that control unit software contains several variants (copies) of some adjustable objects, whereas description file contains only one record to describe. In real application only one variant is in use, depending on car-specific parameters.

VARIANT_CODING

Example:

```
/begin VARIANT_CODING
        VAR_SEPARATOR                   "."             /* PUMKF.1 */
        VAR_NAMING                      NUMERIC

        /* variant criterion "Car body" with three variants */
        /begin VAR_CRITERION            Car  "Car body"  Limousine  Kombi  Cabrio
        /end VAR_CRITERION
        /* variant criterion "Type of gear box" with two variants */
        /begin VAR_CRITERION            Gear  "Type of gear box"  Manual  Automatic
        /end VAR_CRITERION

        /begin VAR_FORBIDDEN_COMB               /* forbidden: Limousine - Manual */
                                        Car Limousine
                                        Gear Manual
        /end VAR_FORBIDDEN_COMB
        /begin VAR_FORBIDDEN_COMB               /* forbidden: Cabrio - Automatic */
                                        Car Cabrio
                                        Gear Automatic
        /end VAR_FORBIDDEN_COMB
        /begin VAR_CHARACTERISTIC  PUMKF  /* define PUMKF as variant coded */
                                        Gear            /* Gear box variants */
                                        /begin VAR_ADDRESS
                                                        0x7140  0x7168
                                        /end VAR_ADDRESS
        /end VAR_CHARACTERISTIC
        /begin VAR_CHARACTERISTIC  NLLM     /* define NLLM as variant coded */
                                        Gear  Car       /* car body and gear box variants */
                                        /begin VAR_ADDRESS
                                                        0x8840  0x8858  0x8870  0x 8888
                                        /end VAR_ADDRESS
        /end VAR_CHARACTERISTIC
/end VARIANT_CODING
```

Type of gear box **Car body**	**MANUAL**	**AUTOMATIC**
Limousine	doesn't exist	NLLM.3 (address = 0x8870)
Kombi	NLLM.1 (address = 0x8840)	NLLM.4 (address = 0x8888)
Cabrio	NLLM.2 (address = 0x8858)	doesn't exist

Type of gear box	
MANUAL	**AUTOMATIC**
PUMKF.0 (address = 0x7140)	PUMKF.1 (address = 0x7168)

Table 5: Example of NLLM and PUMKF - variants coding

6.3.123 VAR_ADDRESS

Prototype:

```
/begin VAR_ADDRESS          { Address }*
/end VAR_ADDRESS
```

Parameters:

long Address: Start address of one variant of variant coded adjustable object.

Description:

This keyword can be used to define a list of start addresses of variant coded adjustable objects (see keyword VAR_CHARACTERISTIC). The number of addresses agrees with number of valid combinations of adjustable objects variant criteria (forbidden combinations excluded). The order of addresses corresponds to the order of variant criteria defined with parameter 'CriterionName' at keyword VAR_CHARACTERISTIC. The priority of index increment is according to the following rules:

- the priority of index increment is invers to the order of variant criteria definition at keyword VAR_CHARACTERISTIC, e.g.:
  - the first variant criterion has the lowest priority
  - the last variant criterion has the highest priority

The following example describes the order of addresses of an adjustable object depending on three variant criterions with 'L', 'N', and 'M' criterion values:
Example:

Crit1 = { $\mathrm{Val}_{1,1}$, $\mathrm{Val}_{1,2}$, ...$\mathrm{Val}_{1,L}$)
Crit2 = { $\mathrm{Val}_{2,1}$, $\mathrm{Val}_{2,2}$, ...$\mathrm{Val}_{2,M}$)
Crit3 = { $\mathrm{Val}_{3,1}$, $\mathrm{Val}_{3,2}$, ...$\mathrm{Val}_{3,N}$)

Corresponding address list:
Address[0] = Address ($\mathrm{Val}_{1,1}$, $\mathrm{Val}_{2,1}$, $\mathrm{Val}_{3,1}$)
Address[1] = Address ($\mathrm{Val}_{1,1}$, $\mathrm{Val}_{2,1}$, $\mathrm{Val}_{3,2}$)
:
Address[N - 1] = Address ($\mathrm{Val}_{1,1}$, $\mathrm{Val}_{2,1}$, $\mathrm{Val}_{3,N}$)
Address[N] = Address ($\mathrm{Val}_{1,1}$, $\mathrm{Val}_{2,2}$, $\mathrm{Val}_{3,1}$)
Address[N + 1] = Address ($\mathrm{Val}_{1,1}$, $\mathrm{Val}_{2,2}$, $\mathrm{Val}_{3,2}$)
:
Address[N + N - 1] = Address ($\mathrm{Val}_{1,1}$, $\mathrm{Val}_{2,2}$, $\mathrm{Val}_{3,N}$)
:

Example:

```
/begin VAR_ADDRESS
        0x8840  0x8858  0x8870  0x 8888               /* see example, page 189 */
/end VAR_ADDRESS
```

6.3.124 VAR_CHARACTERISTIC

Prototype:

```
/begin VAR_CHARACTERISTIC        Name { CriterionName }*
                                 [->VAR_ADDRESS]
/end VAR_CHARACTERISTIC
```

Parameters:

ident Name: Identifier of variant coded adjustable object (refers to CHARACTERISTIC or AXIS_PTS record).

ident CriterionName: Corresponding to each combination of variant criteria defined with this parameter the control unit software contains variants of concerning adjustable object.

Optional Parameters:

->VAR_ADDRESS: Definition of start address of adjustable objects variants.

Description:

This keyword defines one adjustable object to be variant coded, i.e. this adjustable objects is multiple deposited in control unit software corresponding to the assigned variant criteria. The number of variants results on valid combinations (forbidden combinations excluded) of variant criteria.

Example:

```
/begin VAR_CHARACTERISTIC        /* define NLLM as variant coded */
    NLLM
    Gear  Car
    /* gear box including the 2 variants "Manual" and "Automatic" */
    /* car body including the 3 variants "Limousine",  "Kombi" and "Cabrio" */

    /* four addresses corresponding to the four valid combinations */
    /* of criterion 'Gear' and 'Car' (see example, page 189*/
    /begin VAR_ADDRESS    0x8840  0x8858  0x8870  0x 8888
    /end VAR_ADDRESS
/end VAR_CHARACTERISTIC
```

6.3.125 VAR_CRITERION

Prototype:

```
/begin VAR_CRITERION          Name LongIdentifier { Value }*
                              [-> VAR_MEASUREMENT]
/end VAR_CRITERION
```

Parameters:

ident Name:	Identifier of variant criterion.
string LongIdentifier:	Comment to describe the variant criterion.
ident Value:	Enumeration of criterion values.

Optional Parameters:

->VAR_MEASUREMENT: This keyword can be used to specify a special measurement object. This measurement object indicates with its current value the variant which has effect on running control unit software.

Description:

This keyword describes a variant criterion, i.e. some adjustable objects are multiple deposited in control unit software corresponding to the emuneration of variant criterion values.

Example:

```
/* variant criterion "Car body" with three variants */
/begin VAR_CRITERION          Car  "Car body"

                              /* Enumeration of criterion values */
                              Limousine  Kombi  Cabrio

    VAR_MEASUREMENT           S_CAR
/end VAR_CRITERION
```

6.3.126 VAR_FORBIDDEN_COMB

Prototype:

```
/begin VAR_FORBIDDEN_COMB        { CriterionName  CriterionValue }*
/end VAR_FORBIDDEN_COMB
```

Parameters:

ident CriterionName:	Identifier of variant criterion.
ident CriterionValue:	Value of variant criterion ' CriterionName '.

Description:

This keyword describes a forbidden combination of values of different variant criteria.

Example:

```
/* forbidden variant combination (doesn't exist in control unit software): */
/begin VAR_FORBIDDEN_COMB
        Car Limousine           /* variant value 'Limousine' of criterion 'Car'  */
        Gear Manual             /* variant value 'Manual' of criterion 'Gear'  */
/end VAR_FORBIDDEN_COMB
```

6.3.127 VAR_MEASUREMENT

Prototype:

VAR_MEASUREMENT Name

Parameters:

ident Name: Identifier of measurement object which indicates the actual criterion value. This parameter refers to a MEASUREMENT record of description file.

Description:

This keyword can be used to specify a special measurement object. This measurement object indicates with its current value the variant which has effect on running control unit software. The value 0 (zero) of measurement object corresponds to the first variant value defined at relative VAR_CRITERION record (see parameter 'Value' at keyword VAR_CRITERION), the value 1 to the second and so on.

Question:
Which value of measurement object should be used to get the effective variant?
a) internal value acquired from control unit software or?
b) physical value computed from internal value using COMPU_METHOD record?
c) Referenced COMPU_METHOD record could be a 'verbal conversion table' and the strings defined at COMPU_VTAB could correspond to criterion values at VAR_CRITERION record?

```
/begin COMPU_VTAB      V_GEAR_BOX
      "variants of criterion ""Type of Gear Box"""
      3
      17 "Limousine"
      39 "Kombi"
      41 "Cabrio"
/end COMPU_VTAB
```

Note: VAR_MEASUREMENT may only refer to objects of type MEASUREMENT.

Example:

```
/begin VAR_CRITERION               Car  "Car body"  Limousine  Kombi  Cabrio
        VAR_MEASUREMENT            S_GEAR_BOX
/end VAR_CRITERION
/* S_GEAR_BOX = 0:          indicates variant "Limousine" to be effectiv */
/* S_GEAR_BOX = 1:          indicates variant "Kombi" to be effectiv */
/* S_GEAR_BOX = 2:          indicates variant "Cabrio" to be effectiv */
```

6.3.128 VAR_NAMING

Prototype:

VAR_NAMING Tag

Parameters:

enum Tag: Format of variant extension (index). Possible values:

NUMERIC: variant extension is a number (integer: 0,1,2,3...).

This parameter is reserved for future extension (e.g. ALPHA = { A, B, C, D....}).

Description:

This keyword defines the format of variant externsion (index) of adjustable objects name. The extension is used at MCD to distinguish the different variants of adjustable objects.

Example:

```
/* variant extension: see example page 189*/
VAR_NAMING                    NUMERIC
```

6.3.129 VAR_SEPARATOR

Prototype:

VAR_SEPARATOR Separator

Parameters:

String Separator: This parameter defines the separating symbol of variant extension.

Description:

This keyword can be used to define the separating symbol between the two parts of adjustable objects name: 1.) identifier 2.) variant extension.
Remark: The identifier of description record of variant coded adjustable objects contains no variant extension. The extension is needed to distinguish the variants at MCD.

Example:

```
VAR_SEPARATOR          "."              /* example: "PUMKF.1" */
/* three parts of variant coded adjustable objects name:   */
/* 1.) Identifier of adjustable object:   "PUMKF"          */
/* 2.) Separator:                         "." (decimal point)   */
/* 3.) Variants extension:                "1"              */
```

6.3.130 VERSION

Prototype:

VERSION	VersionIdentifier

Parameters:

string VersionIdentifier: short identifier for the version

Description:

String for identification of the version with maximum MAX_LEN (at present MAX_LEN = 100) characters.

Example:

```
VERSION                    "BG5.0815"
```

6.3.131 VIRTUAL

Prototype:

```
/begin VIRTUAL              (MeasuringChannel) *
/end VIRTUAL
```

Parameters:

ident MeasuringChannel: Reference to a measurement (MEASUREMENT) or a virtual measurement (MEASUREMENT, VIRTUAL)

Description:

This keyword allows virtual measurements to be specified. For this, constants, measurements and virtual measurements can be combined into one quantity. The list specified with the VIRTUAL keyword indicates the quantities to be linked (reference). These quantities are combined into one measurement by means of a single conversion formula. The conversion formula must be capable of processing several input quantities.

Example:

```
/begin MEASUREMENT          PHI_FIRING          /* Name */
                            "Firing angle"      /* Long identifier */
                            UWORD               /* Data type */
                            R_PHI_FIRING        /* Conversion */
                            1                   /* Resolution */
                            0.01                /* Accuracy */
                            120.0               /* Lower limit */
                            8400.0              /* Upper limit */

                            /* Quantities to be linked: 2 measurements */
      /begin VIRTUAL        PHI_BASIS  PHI_CORR
      /end VIRTUAL
/end MEASUREMENT

/begin COMPU_METHOD         R_PHI_FIRING        /* Name */
                            "Addition of two measurements"
                            FORM                /* Convers_type */
                            "%4.2"              /* Display format */
                            "GRAD_CS"           /* physical unit */
      /begin FORMULA        "X1 + X2"           /* X1 -> PHI_BASIS */
                                                /* X2 -> PHI_CORR */
      /end FORMULA
/end COMPU_METHOD
```

6.3.132 VIRTUAL_CHARACTERISTIC

Prototype:

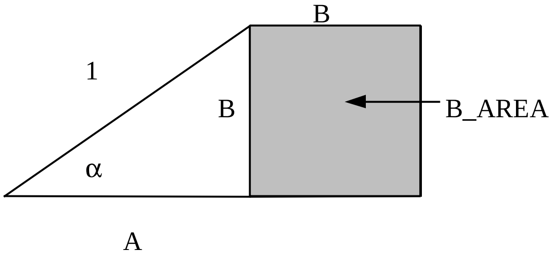
/begin VIRTUAL_CHARACTERISTIC Formula (Characteristic)*
/end VIRTUAL_CHARACTERISTIC

Parameters:

string Formula	Formula to be used for the calculation of characteristic from the value of other characteristics
ident Characteristic	Identifier of those adjustable objects that are used for the calculation of this characteristic.

Description:

This keyword allows to define characteristics that are not deposited in the memory of the control unit, but can be used to indirectly calibrate other characteristic values in the control unit, if these are declared to be dependent on this characteristic. The introduction of virtual characteristic is therefore useful for saving memory in the case the calibration with dependent characteristics is used.



For the initial value of the virtual characteristic must be derived from the values of other characteristics. The mechansim to implement this is the same as for dependent characteristics by a list of characteristics and a formula, e.g. α = arcsin(B). Also B might be virtual, i.e. its value has to be derived from B_AREA.

The following example makes clear how the calibration process takes place. When the virtual characteristic α is initialized, the value of α is calculated from the value of B. Therefore B_{hex} is read from the ECU and $B_{phys} = B_{hex}/100$ is computed. Assuming the value $B_{hex} = 80$, $B_{phys} = 0.8$ and $\alpha_{phys} = \arcsin(B_{phys}) = 53.13$. Since virtual characteristics are not in the memory of an ECU, α_{hex} and α_{phys} may coincide if the datatype for α_{hex} is choosen an float datatype and the conversion formula is the identity (one to one formula).

Used in CHARACTERISTIC.

Example:

```
/begin VIRTUAL_CHARACTERISTIC
        „sin(X1)“
        B
/end VIRTUAL_CHARACTERISTIC
```

7 Include mechanism

7.1 Description of complex projects

For the description of projects involving several control units or application devices of various manufacturers the Include statement can be used.

/include <filename>

This statement allows several description files to be integrated into one project description. The filename may be put between quotation marks. If the filename contains spaces the quotation marks are required.

<u>Example:</u>

```
________________________________________ File PROJECT1.A2L ________________________________

/begin PROJECT                              RAPE-SEED ENGINE "Engine tuning for operation with rape oil"
   /begin HEADER                            "General project description"
      VERSION                               "0815"
      PROJECT_NO               1188
   /end

   /include ENG_ECU.A2L
   /include ABS_ECU.A2L
   /include ″SPEC_ECU.A2L“
/end
________________________________________ End of file PROJECT1.A2L __________________________
```

7.1.1 Description of interface-specific parameters

For parameterization of the drivers and for access to the adjustable and measurement objects different parameters have to be used within the various drivers. The user interface, which does not need to know these parameters, in fact only requires a description of the data types to be able to read in these interface-specific parameters. This description, based on the ASAP2 metalanguage described hereafter, occurs either INLINE within the description file, or in separate files. In the second case the Include statement can be used to integrate the description of interface- specific parameters:

```
/begin MODULE ...
        /include <filename>
        ....
/end MODULE
```

PART C: ASAP2 METALANGUAGE

8 Interface-specific description data

Between the control part of the standardised application system and the program parts for access to the application interface an interface (ASAP 1b) has been defined. The program parts for access to the application interface shall in future be realised as linkable drivers.

Furthermore, the description data in the application system are divided into two categories:

1) Parameters that are used by the control interface.

2) Parameters that are only analysed by the driver and whose meaning is hidden to the control interface (interface-specific parameters). They are transferred to the driver as a binary block.

These two measures should make it possible that new interface module types can be handled without having to introduce any changes in the control part of the application system but simply by incoporating a new driver.

For the description of the *interface-specific parameters* a description language (ASAP2 metalanguage, in short **A2ML**) will be defined on the following pages. Each manufacturer can specify a special set of parameters for their own interface module types (format description). Using this format description (in A2ML) the standardised application system must be capable of reading in the *interface-specific parameters* of the description file and transferring them to the drivers (see Figure 10).

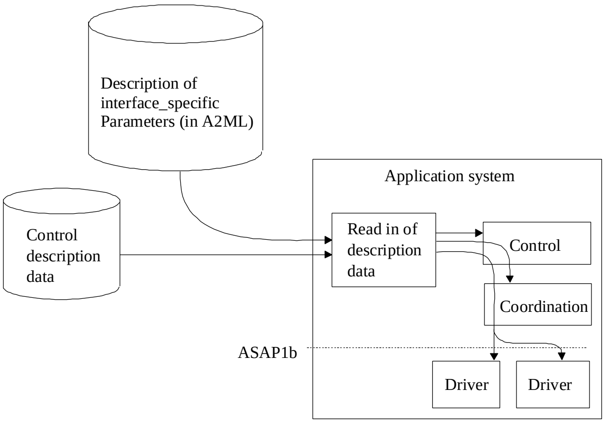


Figure 10: Schematic data flow of description data

8.1 Format of the ASAP2 metalanguage

To describe the grammar of the *ASAP2 metalanguage*, an extended Backus-Naur format is used:

– A non-terminal is represented as a simple identifier:

```
block_definition
```

– The symbol used as inference symbol in the production rules is '::='

```
type_definition ::= type_name
```

– A terminal (keyword) is enclosed within quotes:

```
"struct"
```

– Non-formally defined parts are enclosed within angle brackets. In the description given hereafter the following two identifiers are used in particular:

<tag>:	<tag> is used to define the keywords of the ASAP2 description file by means of a character sequence enclosed within double inverted commas.
<identifier>:	identifier for the definition of data structures.

– An optional part is enclosed within square brackets:

```
[ <identifier> ]
```

– Alternative parts are separated by '|' :

```
"char" | "int" | "long"
```

– Explanations are enclosed within comment symbols:

```
/* comment */
```

8.1.1 Grammar in the extended Backus-Naur format

```
declaration              ::= type_definition ";" | block_definition ";"
```

Definition of a data structure to be used for defining a data record of the description file.

```
type_definition          ::=  type_name

type_name                ::=  predefined_type_name  |
                              struct_type_name  |
                              taggedstruct_type_name  |
                              taggedunion_type_name  |
                              enum_type_name

predefined_type_name    ::=   "char" |
                              "int" |
                              "long" |
                              "uchar" |
                              "uint" |
                              "ulong" |
                              "double" |
                              "float"
```

Definition of a block. A block consists of a special begin keyword ("/begin"), a keyword identifying the record type (e.g. "FUNCTION_LIST"), the relevant data record and an end keyword ("/end"). Nested blocks are also possible.

```
block_definition        ::=   "block" <tag>  type_name
```

Definition of an enumeration:

```
enum_type_name          ::=   "enum" [ <identifier> ] "{" enumerator_list "}"  |
                              "enum" <identifier>

enumerator_list         ::=   enumerator  |
                              enumerator "," enumerator_list

enumerator              ::=   <keyword> [ "=" <constant> ]
```

Definition of data records of the ASAP2 description file with fixed sequence of the data record elements.

```
struct_type_name        ::=   "struct" [ <identifier> ] "{" [ struct_member_list ] "}"  |
                              "struct" <identifier>

struct_member_list      ::=   struct_member |
                              struct_member struct_member_list

struct_member           ::=   member ";"

member                  ::=   type_name [array_ s pecifier]

array_specifier         ::=   "[" <constant> "]"   |
                              "[" <constant> "]" array_specifier
```

Definition of data records of the ASAP2 description file whose elements can specified in a random sequence. All elements are optional and each element is identified by its tag.

For the description of lists with a variable number of elements, the symbols "(" and ")*" are used. The sequences identified by these symbols can be repeated any number of times.

```
taggedstruct_type_name  ::=   "taggedstruct" [ <identifier> ]
                                    "{" [ taggedstruct_member_list ] "}"  |
                              "taggedstruct" <identifier>
```

```
taggedstruct_member_list ::=  taggedstruct_member  |
                              taggedstruct_member  taggedstruct_member_list

taggedstruct_member      ::=  taggedstruct_definition ";"  |
                              "(" taggedstruct_definition ")* ;"  |
                              block_definition ";"  |
                              "(" block_definition ")* ;"

taggedstruct_definition  ::=  <tag>  member  |
                              <tag> "(" member ")*"
```

Definition of variants in data records of the ASAP2 description file. Similar to the 'union' data type used in programming language C, the ASAP2 description file allows only one variant to be specified at a time in a 'taggedunion'. Each variant is assigned a tag for identification purposes (see <tag>).

```
taggedunion_type_name    ::=  "taggedunion" [ <identifier> ]
                                      "{" [ taggedunion_member_list ] "}"  |
                              "taggedunion" <identifier>

taggedunion_member_list  ::=  tagged_union_member  |
                              tagged_union_member taggedunion_member_list

taggedunion_member       ::=  <tag> [ member ] ";" |
                              block_definition ";"
```

8.2 Designing A2ML-file

This chapter is describing how to design an A2ML-file for interface-specific data used together with a ASAP1b compatible driver.

To be compatible with ASAP2 V1.2 a tag ”IF_DATA” should be defined in the A2ML-file. This rag is then used by the MCD to interpret the data that is written in the various IF_DATA-fields in the ASAP2-file. Figure 12 shows a template that preferably should be used to design a A2ML-file for interface-specific data.

The string ”ASAP1B_*” in the beginning of Figure 12 is a tag which is used to separate different interfaces. This makes it possible to define different types of data for different interfaces. The *-character in the template could be substituted with a string that describes the interface (e.g. ASAP1B_RS232 for a serial interface).

The tag SOURCE is described by the ASAP2 specification but because it is located in the IF_DATA it should be defined in the A2ML-file.

The different BLOB tags are used to define interface dependent data that should be put into BLOB-objects and sent to the ASAP1b-driver when different services are called. Figure 11 shows which BLOB-objects that is used by the various ASAP1b-services. The ASAP1b-services missing in the figure does not use any BLOB-objects.

ASAP1b service	BLOB-objects used
INIT_READ	TP_BLOB, QP_BLOB and KP_BLOB
INIT_ACCESS	TP_BLOB and DP_BLOB

ACCESS	PA_BLOB
COMMAND	TP_BLOB

Figure 11: Description of BLOB-objects used in ASAP1b-services

There are two interfaces already defined in the ASAP2 documentation, ASAP1B_CAN and ASAP1B_ADRESS, these interfaces define a minimum of information that could be used to reference measurement parameters inside an ECU respectively on a CAN-network. The A2ML-file for these interfaces is displayed in Figure 13.

```
/begin A2ML
/* template.aml ***************************************************/
/* By: Volvo Car Corporation, 961011                                    */
/*                                                                  */
/* Template for designing IF_DATA fields for ASAP2 files and BLOB's     */
/* for ASAP1b interface.                                                */
/* ************************************************************/
block "IF_DATA" taggedunion if_data
{
    "ASAP1B_*"              /* The tag of ASAP1b specific information should          */
                            /* start with ASAP1B_ then the * character can be         */
                            /* substituted with a name of manufacturer's choice.      */

    taggedstruct            /* optional parameters  */

        (block "SOURCE" struct
        {
            struct          /*  indispensable */
            {
                char [101]; /* source name (string)*/
                int;        /* min period ( conforming together with min factor */
                            /* the fastest samplingrate available ).          */
                long;       /* min factor */
            };
            taggedstruct    /* optional parameters  */
            {
                block "QP_BLOB" struct              /* QP_BLOB for ASAP1b */
                {
                    /* QP_BLOB specification  */
                };
            };
        }
        )*;  /*  multiple SOURCE may exist */

        block "TP_BLOB" struct                      /* TP_BLOB for ASAP1b */
        {
            /* TP_BLOB specification  */
        };

        block "DP_BLOB" struct                      /* DP_BLOB for ASAP1b */
        {
            /* DP_BLOB specification  */
        };

        block "PA_BLOB" struct                      /* PA_BLOB for ASAP1b */
        {
            /* PA_BLOB specification  */
        };

        block "KP_BLOB" struct                      /* KP_BLOB for ASAP1b */
        {
            /* KP_BLOB specification  */
        };

        /*  for MODULE                  may only TP_BLOB and SOURCE be specified                  */
        /*  for CHARACTERISTIC          may only DP_BLOB and PA_BLOB be specified                 */
        /*  for AXIS_PTS          may only DP_BLOB and PA_BLOB be specified              */
        /*  for MEMORY_LAYOUT           may only DP_BLOB and PA_BLOB be specified                 */
        /*  for MEASUREMENT             may only KP_BLOB, DP_BLOB and PA_BLOB be specified        */
    };

    /* Extra tags can be defined here */

};

/***********************************************************************/
/end A2ML
```

Figure 12: Template for A2ML-file

```
/begin A2ML
/* asap2std.aml ********************************************************** */
/* By: Volvo Car Corporation, 961217                                                         */
/*                                                                                   */
/* A2ML-file defining the interfaces ASAP1B_ADDRESS and ASAP1B_CAN.                 */
/* ************************************************************************ */
block "IF_DATA" taggedunion if_data
{
    "ASAP1B_ADDRESS" taggedstruct       /* optional parameters  */
    {
        (block "SOURCE" struct
        {
            struct                          /*  indispensable */
            {
                char [101];                 /* source name (string)*/
                int;                        /* min period ( conforming together with min factor the fastest */
                                            /* ...samplingrate available ). */
                long;               /* min factor */
            };
        }
        )*;                                 /*  multiple SOURCE may exist */

        block "KP_BLOB" struct              /* KP_BLOB specification for ASAP1b */
        {
            long;                           /* address of measurement object (e.g. emulator RAM addressing) */
        };
    };

    "ASAP1B_CAN"  taggedstruct              /* optional parameters  */
    {
        (block "SOURCE" struct
        {
            struct                          /*  indispensable */
            {
                char [101];                 /* source name (string)*/
                int;                        /* min period ( conforming together with min factor */
                                            /* the fastest samplingrate available ).           */
                long;                       /* min factor */
            };
        }
        )*;                                 /*  multiple SOURCE may exist */

        block "KP_BLOB" struct              /* KP_BLOB specification for ASAP1b */
        {
            char[33];                       /* messagename */
            long;                   /* identifier */
            int;                            /* messagesize */
            char[101];                      /* messagesource */
            int;                            /* startbit */
            int;                            /* datasize */
            taggedstruct
            {
                ”MULTIPLEX” struct
                {
                    int;                    /* startbit */
                    int;                    /* datasize */
                    long;                   /* tag */
                };
            };
        };                                  /* end of: block "KP_BLOB" */
    };                                      /* end of: "ASAP1B_CAN"  taggedstruct */
};                                          /* end of:  taggedunion if_data */

/***********************************************************************/
/end A2ML
```

Figure 13: A2ML-file for ASAP2 standard interfaces

8.3 Structure of BLOBs

As described at ASAP1b specification the ASAP1b device needs some 'binary large objects' (BLOBs) containing interface depending parameters. As long as the structure of BLOB's doesn't contain any optional parts or any parameter arrays with variable length, the structure of BLOBs at ASAP1b device need no further explanation:

- The structure of BLOBs is equivalent to the structure described at A2ML-file (also see "6.2 Predefined data types").

Since ASAP2 Version 1.31 there is a specification for the storage layout of the BLOBs which also describes BLOBs with optional parts.

8.4 Example of ASAP2 metalanguage

The following example illustrates the ASAP2 metalanguage (A2ML) on the basis of the format definition of ASAP2 version 1.0. It is not adjusted to the modifications of version 1.20 and version 1.21.

Remark:
The ASAP2 metalanguage shall be used to specify the format of the interface-specific parameters. The following format specification for the complete ASAP2 description file is only intended as an **example** to illustrate the ASAP2 metalanguage.

```
_________________________________ File EXAMPLE.AML ___________________________________________
block  "PROJECT"  struct project;

struct project (
  struct (                                              /* mandatory part */
     char[20];                                          /* name */
     char[100];                                         /* long identifier */
  }
  taggedstruct {                                        /* optional part */
     block "HEADER" struct projectheader;
     ( block "MODULE" struct modul )*;           /* block "MODULE" can occur several times */
  }
};

struct modul {
   struct                                               /* mandatory part */
     char[20];                                          /* name */
     char[100];                                         /* description */
  }
  taggedstruct {                                        /* optional part */
     block  "MOD_PAR"            struct mod_par;
     block  "MOD_COMMON"                  struct mod_common;

     /* taggedunion interface_param: to be specified by manufacturer */
     ( block  "IF DATA"                   taggedunion interface_param )*;
     ( block  "CHARACTERISTIC"            struct characteristic )*;          /* (0...n) times  */
     ( block  "AXIS_PTS"                  struct axis_pts )*;
     ( block  "MEASUREMENT"               struct measurement )*;
     ( block  "COMPU_METHOD"              struct conversion )*;
     ( block  "COMPU_TAB"                 struct conv_tab )*;
     ( block  "COMPU_VTAB"                struct conv_tab_verb )*;
     ( block  "FUNCTION"         struct function )*;
     ( block  "RECORD_LAYOUT"             taggedstruct record_layout )*;
  };
};
```

```
/* ******************************** project header **************************** */
struct projectheader {
   char[100];                                          /* comment */
   taggedstruct {
      "VERSION"          char[100];
      "PROJECT_NO"       char[100];
   };
};

/* ***************************** module description *************************** */
struct memory_loc {
   enum prg_type { "PRG_CODE" = 0, "PRG_DATA" = 1, "PRG_RESERVED = 2 };     /* prg_type */
   long;                                               /* addr */
   long;                                               /* size */
   long[5]                                             /* offset */
};

struct sdk {
   char[20]                                            /* label_name */
   char[20]                                            /* value */
};

struct mod_par {
   char[100];                                          /* comment */
   taggedstruct {                              /* optional part */
      "VERSION"                char[100];              /* data status */
      "ADD_EPK"                long;
      "EPK"                    char[100];
      "SUPPLIER"               char[100];
      "CUSOTMER"               char[100];
      "CUSTOMER_NO"            char[40];
      "USER"                   char[40];               /* applications engineer */
      "PHONE_NO"               char[40];
      "ECU"                    char[40];
      "CPU_TYPE"               char[40];
      "NO_OF_INTERFACES"       uint;
      ( "MEMORY_LAYOUT"        struct memory_loc )*;
      ( "SYSTEM_CONSTANT" struct sdk )*;
   };
};

enum endian { "BIG_ENDIAN" = 2, "LITTLE_ENDIAN" = 1 };

enum datatype { "UBYTE" = 0, "SBYTE" = 1, "UWORD" = 2, "SWORD" = 3, "ULONG" = 4, "SLONG" = 5 };

/* ************************ control unit description ************************ */
struct mod_common {
   char[100];                                          /* comment */
   taggedstruct {
      "S_REC_LAYOUT"           char[20];
      "DEPOSIT"                enum {  "DIFFERENCE" = 0,  "ABSOLUTE" = 1 };
      "BYTE_ORDER"             enum endian;
      "DATA_SIZE"              int;                    /* 8/16 bit */
   };
};
```

```
/* ********************** description of characteristics ********************** */
struct axis_descr {
   struct {                                                    /* mandatory part */
      enum { "STD_AXIS" = 0, "COM_AXIS" = 1, "FIX_AXIS" = 2 };
      char[20];                                                /* input quantity */
      char[20];                                                /* conversion */
      int;                                                     /* maximum number of axis points */
      double;                                                  /* lower limit */
      double;                                                  /* upper limit */
   };
   taggedstruct {                                              /* optional part */
      "MAX_GRAD" double;                                       /* maximum gradient */
      "MONOTONY" enum { "MON_INCREASE" = 0, "MON_DECREASE" = 1 }; /* monotony */
      "BYTE_ORDER" enum endian;                                /* byte order */
      "FIX_AXIS_PAR" struct {                                  /* parameters of fixed characteristic curve/map */
         int;                                                  /* offset */
         int;                                                  /* shift */
         int;                                                  /* number of axis points */
      };
      "DEPOSIT" enum { "DIFFERENCE" = 0,  "ABSOLUTE" = 1 };  /* with deviation from standard */
      "AXIS_PTS_REF" char[20];                                 /* reference to axis points distribution */
   };
};


struct characteristic {
   struct {                                                    /* mandatory part */
      char[20];                                                /* name      */
      char[120];                                               /* descr     */
      enum { "VALUE"=0, "CURVE"=1, "MAP"=2, "CUBOID"=3, "VAL_BLK"=4, "ASCII"=5};    /* type */
      long;                                                    /* address   */
      char[20];                                                /* deposit */
      double;                                                  /* maxdiff */
      char[20];                                                /* conversion */
      double;                                                  /* lower limit */
      double;                                                   /* upper limit */
   };
   taggedstruct {                                              /* optional part */
      "BYTE_ORDER" enum endian;                                /* byte order */
      "BIT_MASK"  long;                                        /* bit mask */
      "FUNCTION_LIST" ( char[20] )*;                           /* functions */
      "NUMBER" int;                                 /* number of constants or characters */
      "EXTENDED_LIMITS" struct {
         double;                                               /* extended lower limit */
         double;                                               /* extended upper limit */
       };
      ( block "AXIS_DESCR" struct axis_descr )*;           /* axis description */
      ( "IF_DATA" taggedunion ifp_characteristic )*;       /* interface-specific part */
   };          /* taggedunion ifp_characteristic: to be specified by manufacturer */
};

/* **************** Description of axis point distributions ******************  */
struct axis_pts {
   struct {                                                    /* mandatory part */
      char[20];                                                /* name      */
      char[120];                                               /* descr     */
      long;                                                    /* address   */
      char[20];                                                /* input quantity */
      char[20];                                                /* deposit */
      double;                                                  /* maxdiff */
      char[20];                                                /* conversion */
      int;                                                     /* maximum number of axis points */
      double;                                                  /* lower limit */
      double;                                                  /* upper limit */
   };
   taggedstruct {                                              /* optional part */
      "DEPOSIT" enum {  "DIFFERENCE" = 0,  "ABSOLUTE" = 1 };
      "BYTE_ORDER" enum endian;                                /* byte order */
      "FUNCTION_LIST" ( char[20] )*;                           /* functions */
      ( "IF_DATA" taggedunion ifp_characteristic )*;       /* interface-specific part */
   };          /* taggedunion ifp_characteristic: to be specified by manufacturer */
};

/* *********************** Description of measurements ***********************  */
struct measurement {
   struct {                                         /* mandatory part */
```

```
    char[20];                                    /* name      */
    char[120];                                             /* descr     */
    enum datatype;                                         /* data type */
    char[20];                                              /* conversion method */
    int;                                         /* resolution in bits */
    double;                                      /* accuracy */
    double;                                      /* lower limit */
    double;                                      /* upper limit */
  };
  taggedstruct {                                           /* optional part */
    "BIT_MASK"              long;                          /* bit mask */
    "BYTE_ORDER"            enum endian;                   /* byte order */
    "FUNCTION_LIST"         ( char[20] )*;                 /* functions: keyword only once,
                                                           /* followed by list of function names */
    "MAX_REFRESH    struct {
      double;                                    /* value */
      enum { "MSEC" = 0, "GRAD_CS" = 1 };                  /* value in 'msec' or in 'crankshaft grad' */
    };
    "VIRTUAL"         ( char[20] )*;             /* list of measurements to be linked */
          /* taggedunion ifp_measurement: to be specified by manufacturer */
    ( "IF_DATA" taggedunion ifp_measurement )*;  /* interface-specific part */
  };
};

/* ********************* Description of conversion method ******************* */
struct conversion {
  struct {                                                           /* mandatory part */
    char[20];                                                        /* name      */
    char[120];                                                       /* descr     */
    enum { "TAB_INTP" = 0, "TAB_NOINTP" = 1, "RAT_FUNC" = 2, "FORM" = 3};     /* type */
    char[30];                                                        /* format (number of digits or fractional digits) */
    char[30];                                                        /* unit        */
  }
  taggedstruct {
    "COEFFS" double[6];                                              /* coefficients  */
    "COMPU_TAB" char [20]                                  /* reference to table */
    "FORMULA" char [100]                                             /* formula */
  }
};

/* ******************* Description of conversion tables ******************** */
struct conv_tab {
  char[20];                                                          /* name          */
  char[120];                                                         /* descr     */
  enum { "TAB_INTP" = 0, "TAB_NOINTP" = 1};                /* type        */
  int;                                                               /* value_pairs_no */
  struct tab {
    long;                                                            /* int_val        */
    double;                                                          /* phys_val       */
  } [256];                                                           /* tab            */
};

/* ************* Description of verbal conversion tables ************* */
struct conv_tab_verb {
  char[20];                                                          /* name          */
  char[120];                                                         /* descr     */
  int;                                                               /* type          */
  int;                                                               /* value_pairs_no */
  struct tab {
    long;                                                            /* int_val        */
    char[40];                                                        /* text           */
  } [20];
};
```

```
/* ************* Description of functions ************** */
struct function {
   char[20];                                                      /* name      */
   char[120];                                                     /* descr     */
};


/* ********************** Description of record layouts ****************** */
enum addrtype { "PBYTE" = 1, "PWORD" = 2, "PLONG" = 3, "DIRECT" = 4 };

struct fnc_values {
   int no;
   enum datatype;                                         /* data type of table values */
   enum { "COLUMN_DIR" = 1, "ROW_DIR" = 2 };              /* row or column oriented */
   enum addrtype;                                         /* addressing */
};

struct axis_pts {
   int no;
   enum datatype;                                         /* data type of axis points */
   enum { "INDEX_INCR" = 1, "INDEX_DECR" = 2 };           /* increasing, decreasing index */
   enum addrtype;                                         /* addressing */
};

struct abl_addr {
   int no;
};

struct abl_datatype {
   int no;
   enum datatype;                                         /* data type */
};
struct record_layout {
   struct {                                                       /* optional part */
      char[20];                                                   /* name */
   };
   taggedstruct {                                                 /* mandatory part */
      "FNC_VALUES"            struct fnc_values;
      "IDENTIFICATION"        struct abl_datatype;
      ("RESERVED"             struct abl_datatypeabl_) *;
      "AXIS_PTS_X"            struct axis_pts;
      "AXIS_PTS_Y"            struct axis_pts;
      "NO_AXIS_PTS_X"         struct abl_datatypeabl_;
      "NO_AXIS_PTS_Y"         struct abl_datatypeabl_;
      "FIX_NO_AXIS_PTS_X"     int;                                /* number of axis points */
      "FIX_NO_AXIS_PTS_Y"     int;                                /* number of axis points */
      "SCR_ADDR_X"            struct abl_addr;
      "SCR_ADDR_Y"            struct abl_addr;
      "RIP_ADDR_X"            struct abl_addr;
      "RIP_ADDR_Y"            struct abl_addr;
      "SHIFT_OP_X"            struct abl_datatypeabl_;
      "SHIFT_        OP_Y"            struct abl_datatypeabl_;
      "OFFSET_X"              struct abl_datatypeabl_;
      "OFFSET_Y"              struct abl_datatypeabl_;
   };
};

______________________________  End File EXAMPLE.AML  ____________________________________
```

8.5 Example of description file

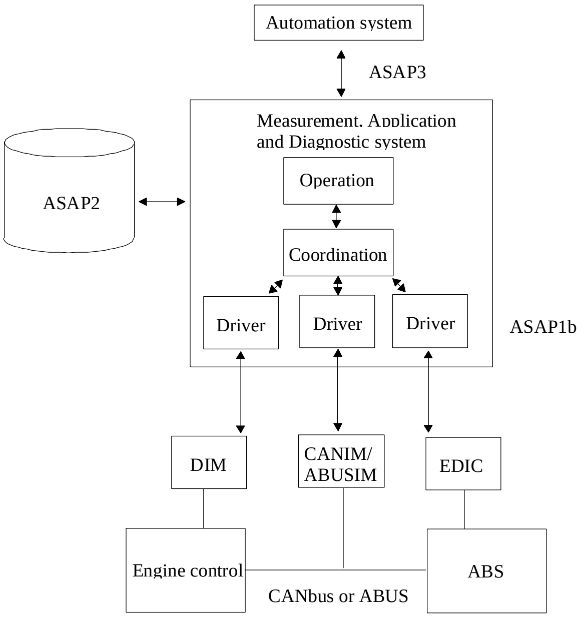


Figure 14: Project example

On the basis of the example shown in Figure 14 it is assumed that the drivers for DIM and CANIM are supplied by supplier 1 and the driver for EDIC by supplier 2. Both suppliers must then specify the formats for the interface-specific parameters. This is done below in the two files SUPP1_IF.AML and SUPP2.IF.AML.

```
________________________________________ File SUPP1_IF.AML ________________________________________
/begin A2ML
/* A2ML-file defining the interfaces ASAP1B_DIM and ASAP1B_CAN.          */
/* ********************************************************************** */

enum mem_typ {  "INTERN" = 0,  "EXTERN" = 1 };
enum addr_typ {  "BYTE" = 1,  "WORD" = 2,  "LONG" = 4 };
enum addr_mode { "DIRECT" = 0, "INDIRECT" = 1 };

taggedunion if_data {
    "ASAP1B_DIM" taggedstruct {             /* optional parameters  */
        (block "SOURCE" struct  {
            struct {                        /*  indispensable */
                char [101];                 /* source name (string)*/
                int;                        /* min period ( conforming together with min factor the fastest */
                                            /* ...samplingrate available ). */
                long;             /* min factor */
            };
            taggedstruct {
                block "QP_BLOB" struct  {
                    long;                   /* adr_distab */
                    int;                    /* len_distab */
                    long;                   /* addr_outp  */
                    long;                   /* trgid      */
                };
            };
        }
        )*;                                 /* multiple SOURCE */
        block "TP_BLOB" struct  {
            int;                            /* display table type */
        };

        block "KP_BLOB" struct {            /* KP_BLOB specification for ASAP1b */
            long;                           /* address   */
            enum addr_typ;                  /* addr_size */
        };

        block "DP_BLOB" struct {            /* DP_BLOB specification for ASAP1b */
            enum mem_typ;                   /* mem_typ   */
        };
        block "PA_BLOB" struct {            /* PA_BLOB specification for ASAP1b */
            enum addr_mode;                 /* addressing mode */
        };
    };                                      /* end of: "ASAP1B_DIM  taggedstruct */

    "ASAP1B_CAN"  taggedstruct {            /* optional parameters  */
        (block "SOURCE" struct {
            struct {                        /*  indispensable */
                char [101];                 /* source name (string)*/
                int;                        /* min period ( conforming together with min factor */
                                            /* the fastest samplingrate available ).           */
                long;                       /* min factor */
            };
        };
        )*;                                 /*  multiple SOURCE may exist */

        block "TP_BLOB" struct {
            int;                            /* bus timing */
        };

        block "KP_BLOB" struct {            /* KP_BLOB specification for ASAP1b */
            char[33];                       /* messagename */
            long;                   /* identifier */
            int;                            /* messagesize */
            char[101];                      /* messagesource */
            int;                            /* startbit */
            int;                            /* datasize */
            taggedstruct {
                ”MULTIPLEX” struct {
                    int;                    /* startbit */
                    int;                    /* datasize */
                    long;                   /* tag */
                };
            };
        };                                  /* end of: block "KP_BLOB" */
    };                                      /* end of: "ASAP1B_CAN"  taggedstruct */
};                                          /* end of:  taggedunion if_data */
/end A2ML
___________________________________________ End of file SUPP1_IF.AML ____________________________________


___________________________________________ File SUPP2_IF.AML ___________________________________________
/begin A2ML
taggedunion if_data {
    "ASAP1B_EDIC" taggedstruct {            /* optional parameters  */
        (block "SOURCE" struct {
            struct {                        /*  indispensable */
                char [101];                 /* source name (string)*/
                int;                        /* min period ( conforming together with min factor the fastest */
                                            /* ...samplingrate available ). */
                long;               /* min factor */
            };
            taggedstruct {
                block "QP_BLOB" struct {
                    :
                };
            };
        }
        )*;                                 /* multiple SOURCE */

        block "TP_BLOB" struct  {
            :
        };
```

```
            block "KP_BLOB" struct {            /* KP_BLOB specification for ASAP1b */
                 :
            };

            block "DP_BLOB" struct {            /* DP_BLOB specification for ASAP1b */
                 :
            };

            block "PA_BLOB" struct {            /* PA_BLOB specification for ASAP1b */
                 :
            };
       };                                       /* end of: "ASAP1B_EDIC"  taggedstruct */
};                                              /* end of:  taggedunion if_data */
/end A2ML
_______________________________________ end of file SUPP2_IF.AML ___________________________________
```

The ASAP2 description files of both suppliers could look as follows:

```
_______________________________________ File MST_ABS.A2L _________________________________
/begin PROJECT          MST_ABS  "Project example see Figure 14"
   /begin HEADER        "General project description"
      VERSION            "0815"
      PROJECT_NO         1188
   /end HEADER

   /include engine_ecu.a2l
   /include abs_ecu.a2l
/end PROJECT
_______________________________________ end of file MST_ABS.A2L _____________________________


_______________________________________ File ENGINE_ECU.A2L _______________________________
/begin MODULE  DIM  "Comment on module"             /* Detailed description of an application device */

   /include "supp1_if.aml"                                 /* Specification of the interface-specific parts */

   /begin MOD_PAR                         "Comment"
      VERSION                             "Test version 09.11.93"
      ADDR_EPK                             0x12345
      EPK                                  "EPROM identifier test"
      SUPPLIER                             "Mustermann"
      CUSTOMER                             "LANZ-Landmaschinen"
      CUSTOMER_NO                          "0987654321"
      USER                       "Ignaz Lanz"            /* Applications engineer */
      PHONE_NO                             "(01111) 22222"
      ECU                                  "Engine control"
      CPU_TYPE                             "Intel 0815"
      NO_OF_INTERFACES                    2
      /begin MEMORY_LAYOUT                PRG_DATA            0x0000   0x8000  -1  -1  -1  -1  -1
              /begin IF_DATA ASAP1B_DIM
                          /begin DP_BLOB EXTERN /end DP_BLOB /* memory type */
                          /begin PA_BLOB DIRECT /end PA_BLOB             /* addressing mode */
              /end IF_DATA
      /end MEMORY_LAYOUT
      SYSTEM_CONSTANT                     "CONTROLLERx CONSTANT1"   "0.99"
      SYSTEM_CONSTANT                     "CONTROLLERx CONSTANT2"   "2.88"
      SYSTEM_CONSTANT                     "CONTROLLERx CONSTANT3"   "-7"
      SYSTEM_CONSTANT                     "ANY-PARAMETER"          "3.14159"
   /end MOD_PAR

   /begin MOD_COMMON                      "Characteristic maps always deposited in same mode"
      DEPOSIT                             ABSOLUTE
      BYTE_ORDER                          BIG_ENDIAN
      DATA_SIZE                           16                  /* bit */
   /end MOD_COMMON
```

```
/begin IF_DATA ASAP1B_DIM
        /begin SOURCE "angular synchonous" 101 1
                /begin QP_BLOB 0x5661  20   0xE001    2
                /end QP_BLOB
        /end SOURCE
        /begin SOURCE "time synchronous, rate 20ms" 4 2
                /begin QP_BLOB 0x3441  20   0xE041    3
                /end QP_BLOB
        /end SOURCE
        /begin TP_BLOB 14   /end TP_BLOB
/end IF_DATA

/begin IF_DATA ASAP1B_CAN
        /begin SOURCE "observing CAN-objects" 1000  1 /end SOURCE
        /begin TP_BLOB $FA /end TP_BLOB
/end IF_DATA

/begin CHARACTERISTIC     KI "I share for speed limitation"
                                VALUE           /* type: constant */
                                0/408F          /* address */
                                DAMOS_FW        /* deposit */
                                5.0     /* max_diff */
                                FACTOR01        /* conversion */
                                0.0     /* lower limit */
                                255.0           /* upper limit */

        /* interface-spec. parameters: address location, addressing */
        /begin IF_DATA  ASAP1B_DIM
                /begin DP_BLOB EXTERN /end DP_BLOB /* memory type */
                /begin PA_BLOB DIRECT /end PA_BLOB              /* addressing mode */
        /end IF_DATA
        /begin FUNCTION_LIST            V_LIM                   /* reference to functions */
        /end FUNCTION_LIST
/end CHARACTERISTIC

/begin CHARACTERISTIC     PUMCD   "Pump characteristic map"
                                MAP                     /* type: characteristic map
                                0x7140                  /* address */
                                DAMOS_KF                /* deposit */
                                100.0                   /* max_diff */
                                VOLTAGE                 /* conversion */
                                0.0                     /* lower limit */
                                5000.0                  /* upper limit */
                                                        /* interface-spec. parameters: address location, addressing */
        /begin IF_DATA  ASAP1B_DIM
                /begin DP_BLOB EXTERN /end DP_BLOB /* memory type */
                /begin PA_BLOB INDIRECT /end PA_BLOB                    /* addressing mode */
        /end IF_DATA
        /begin AXIS_DESCR                       /* X-axis: */
                                STD_AXIS                /* standard axis (no group or fixed characteristic map) */
                                N                       /* input quantity */
                                N_RULE                  /* conversion */
                                16                      /* maximum number of axis points */
                                0.0                     /* lower limit */
                                5800.0                  /* upper limit */
                MAX_GRAD        20.0                    /* max_grad */
        /end AXIS_DESCR
        /begin FUNCTION_LIST            CLDSTRT  FLLD           /* reference to functions */
        /end FUNCTION_LIST
/end CHARACTERISTIC
```

```
/begin MEASUREMENT    M_ECORR
                              "corrected fuel mass"
                              UWORD                   /* data type */
                              M_E                     /* reference to conversion method */
                              1                       /* resolution in bits */
                              0.001                   /* accuracy in '%' */
                              0.0                     /* lower limit */
                              43.0                    /* upper limit */
      BIT_MASK                0xOFF                   /* bit mask */
      /begin IF_DATA   ASAP1B_DIM
              /begin DP_BLOB EXTERN /end DP_BLOB /* memory type */
              /begin PA_BLOB DIRECT /end PA_BLOB      /* addressing mode */
              /begin KP_BLOB 0x8038 WORD /end KP_BLOB /* address, address length */
      /end IF_DATA
      /begin IF_DATA   ASAP1B_CAN
              /begin KP_BLOB                          /* interface-specific part */
                              "message-x"             /* message name */
                              0x0123                  /* identifier */
                              8                       /* message length */
                              "sender-Y"      /* sender */
                              5                       /* start bit */
                              16                      /* bit length */
              /end KP_BLOB
      /end IF_DATA
      /begin FUNCTION_LIST    CLDSTRT  FLLD           /* reference to functins */
      /end FUNCTION_LIST
/end MEASUREMENT

/begin MEASUREMENT      N
                              "current speed"
                              UWORD                   /* data type */
                              N_RULE                  /* reference to conversion method */
                              4                       /* resolution in bits */
                              0.006                   /* accuracy in '%' */
                              0.0                     /* lower limit */
                              5800.0                  /* upper limit */
      BIT_MASK                0xFFFF                  /* bit mask */
      /begin IF_DATA  ASAP1B_DIM
              /begin DP_BLOB EXTERN /end DP_BLOB /* memory type */
              /begin PA_BLOB DIRECT /end PA_BLOB      /* addressing mode */
              /begin KP_BLOB 0x8020 WORD /end KP_BLOB /* address, address length */
      /end IF_DATA
      /begin IF_DATA   ASAP1B_CAN
              /begin KP_BLOB                  /* interface-specific part */
                              "message-x"             /* message name */
                              0x0123                  /* identifier */
                              8                       /* message length */
                              "sender-Y"      /* sender */
                              21                      /* start bit */
                              16                      /* bit length */
              /end KP_BLOB
      /end IF_DATA
      /begin FUNCTION_LIST   V_LIM  CLDSTRT  FLLD     /* reference to functions */
      /end FUNCTION_LIST
/end MEASUREMENT

/begin COMPU_METHOD     FACTOR01                      /* name */
                              "factor 1"              /* long identifier */
                              RAT_FUNC                /* fractional rational function */
                              "%4.0"                  /* format string */
                              ""                      /* unit */
                                                      /* coefficients for polynome conversion */
      COEFFS                  0.0  1.0  0.0  0.0  1.0  0.0
/end COMPU_METHOD

/begin COMPU_METHOD     M_E                           /* name */
                              "amount"                /* long identifier */
                              TAB_INTP                /* conversion table with interpolation */
                              "%4.0"                  /* format string */
                              "mg/H"                  /* unit */
      COMPU_TAB               "AMOUNT"                /* reference to table */
/end COMPU_METHOD
```

```
/begin COMPU_METHOD        N_RULE                                        /* name /*
                                     "speed"                             /* long identifier */
                                     RAT_FUNC                            /* fractional rational function */
                                     "%4.0"                              /* format string */
                                     "1/min                              /* unit */
                                                                         /* coefficients for polynome conversion: "don't care" */
        COEFFS                       0.0 255.0  0.0  0.0  5800.0  0.0
/end COMPU_METHOD

/begin COMPU_METHOD        VOLTAGE                                       /* name */
                                     "voltage"                           /* long identifier */
                                     RAT_FUNC                            /* fractional rational function */
                                     "%4.0"                              /* format string */
                                     "mV"                                /* unit */
                                                                         /* coefficients for polynome conversion: "don't care" */
        COEFFS                       0.0 255.0  0.0  0.0  5000.0  0.0
/end COMPU_METHOD
/begin COMPU_TAB           AMOUNT                                        /* name */
                                     "conversion table for AMOUNT"
                                     TAB_INTP                            /* table with interpolation */
                                     4                                   /* number of value pairs   */
                                     0  0.0  100  10.0  156  30.0  255  43.0      /* value pairs */
/end COMPU_TAB

/begin FUNCTION                      V_LIM  "speed limitation"           /end FUNCTION
/begin FUNCTION                      CLDSTRT "cold start"                /end FUNCTION
/begin FUNCTION                      FLLD  "full load"                   /end FUNCTION

                                                                         /* BOSCH record layout */
/begin RECORD_LAYOUT       DAMOS_FW                                      /* DAMOS constant */
        FNC_VALUES                                                       /* description of function value: */
                                     1                                   /* position in memory */
                                     UBYTE                               /* data type of the constant */
                                     COLUMN_DIR                          /* deposited in columns (don't care) */
                                     DIRECT                              /* direct addressing */
/end RECORD_LAYOUT

/begin RECORD_LAYOUT       DAMOS_KF                        /* DAMOS characteristic diagram
        SRC_ADDR_X                                         /* description of the addresses of the X-input quantities */
                                     1                                   /* position in memory */
                                     UWORD                               /* datatype */
        NO_AXIS_PTS_X                                      /* description of the number of X-axis points */
                                     2                                   /* position in memory */
                                     UBYTE                               /* word length */
        AXIS_PTS_X                                                       /* description of the X-axis point values */
                                     3                                   /* position in memory */
                                     UBYTE                               /* data type of the axis point values */
                                     INDEX_INCR                          /* increasing index with increasing addresses */
                                     DIRECT                              /* direct addressing */
        SRC_ADDR_Y                                         /* description of the addresses of the Y-input quantities */
                                     4                                   /* position in memory */
                                     UWORD                               /* datatype */
        NO_AXIS_PTS_Y                                      /* description of the number of Y-axis points */
                                     5                                   /* position in memory */
                                     UBYTE                               /* word length */
        AXIS_PTS_Y                                                       /* description of the Y-axis point values */
                                     6                                   /* position in memory */
                                     UBYTE                               /* data type of the axis point values */
                                     INDEX_INCR                          /* increasing index with increasing addresses */
                                     DIRECT                              /* direct addressing */
        FNC_VALUES                                         /* description of the function values */
                                     7                                   /* position in memory */
                                     UBYTE                               /* data type of the table values */
                                     COLUMN_DIR                          /* deposited in columns */
                                     DIRECT                              /* direct addressing */
/end RECORD_LAYOUT
```

```
                                                                                            /* SIEMENS record layout */
  /begin RECORD_LAYOUT     SIEMENS_KF                /* SIEMENS characteristic map */
                                                     /* description of the function values: axis points */
                                                     /* are described in an additional specification */
                                  1                          /* position in memory */
                                  UWORD                      /* data type of the table values */
                                  COLUMN_DIR                 /* deposited in columns */
                                  DIRECT                     /* direct addressing */
  /end RECORD_LAYOUT

  /begin RECORD_LAYOUT     SIEMENS_SST               /* SIEMENS axis points distribution */
         AXIS_PTS_X                                  /* description of the axis point values */
                                  1                          /* position in memory */
                                  UWORD                      /* data type of the axis point values */
                                  INDEX_INCR                 /* increasing index with increasing addresses */
                                  DIRECT                     /* direct addressing */
  /end RECORD_LAYOUT
/end MODULE
___________________________________ end of file ENGINE_ECU.A2L ___________________________________________


__________________________________________ file ABS.ECU.A2L ____________________________________
/begin MODULE  EDIC  "Comment on module"             /* detailed description of an application device  */


  /include "supp2_if_aml"                            /* specification of the interface-specific parts */

  /begin MOD_PAR           "comments"
    VERSION                "test version 09.11.93"
    ECU                           "ABS control"
  /end MOD_PAR

  /begin MOD_COMMON        ”comments on these parameters"
    DEPOSIT                ABSOLUTE
    BYTE_ORDER             BIG_ENDIAN
    DATA_SIZE              16 /* bit */
  /end MOD_COMMON

  /begin IF_DATA ASAP1B_EDIC
         /begin SOURCE
                           :
                 /begin QP_BLOB
                           :
                 /end QP_BLOB
         /end SOURCE
         /begin TP_BLOB
                           :
         /end TP_BLOB
  /begin IF_DATA


  /begin CHARACTERISTIC
                                  :
  /end CHARACTERISTIC
```

```
   /begin MEASUREMENT      N
                           "engine speed"
                           UWORD                    /* data type */
                           R_SPEED_3                /* reference to conversion method */
                           2                        /* resolution in bits */
                           2.5                      /* accuracy in '%' */
                           120.0                    /* lower limit */
                                   8400.0                   /* upper limit */
           BIT_MASK                0x0FFF                   /* bit mask */
           BYTE_ORDER              LITTLE_ENDIAN
           /begin IF_DATA  ASAP1B_EDIC
                   /begin KP_BLOB                           /* interface-specific part */
                           "SND"  0x10  0x00  0x05  0x08
                           "RCV"  4  UWORD
                   /end KP_BLOB
                   /begin PA_BLOB
                                   :
                   /end PA_BLOB
                   /begin DP_BLOB
                                   :
                   /end DP_BLOB
           /end IF_DATA
           /begin FUNCTION_LIST    ID_ADJUSTM  FL_ADJUSTM   /* reference to functions */
           /end FUNCTION_LIST
   /end MEASUREMENT
   :
   :
   :
/end MODULE
__________________________________ end of file ABS_ECU.A2L ___________________________________________-
```

8.6 AML example for ASAP1a-CCP (HP, ETAS, Vector)

Some elements in this example are not compatible with ASAP2 V1.2. Therefore extensions are necessary which lead to new keywords in ASAP2 V1.3
Remarks :

1. In the following modified proposal comments have been translated into English.

2. The example is only valid with ASAP2 V1.3 (new keywords)

```
/****************************************************************************/
/*                                                                          */
/*   ASAP2 Meta Language for CCP CAN Calibration Protocol V2.1              */
/*   Assumes ASAP2 V1.3 or later                                            */
/*                                                                          */
/*   AML Version V2.3, 13.10.1998                                           */
/*                                                                          */
/*   Vector Informatik, Zaiser                                              */
/*   Hewlett Packard, Krueger                                               */
/*   ETAS, Maier                                                            */
/*   SIEMENS Automotive, Stuhler                                            */
/*                                                                          */
/*   Datatypes:                                                             */
/*                                                                          */
/*   A2ML       ASAP2          Windows  Erlaeuuterung                       */
/*   ----------------------------------------------------------------       */
/*   uchar      UBYTE          BYTE     unsigned 8 Bit                      */
/*   char       SBYTE          char     signed 8 Bit                        */
/*   uint       UWORD          WORD     unsigned integer 16 Bit             */
/*   int        SWORD          int      signed integer 16 Bit               */
/*   ulong      ULONG          DWORD    unsigned integer 32 Bit             */
/*   long       SLONG          LONG     signed integer 32 Bit               */
/*   float      FLOAT32_IEEE            float 32 Bit                        */
/*                                                                          */
/****************************************************************************/
block "IF_DATA" taggedunion {

  "ASAP1B_CCP" taggedstruct {

    /* Beschreibung der DAQ-Listen */
    (block "SOURCE" struct {

      struct {
        char [101];   /* Name of the DAQ-List (data acquisition list),
                         measurement source .        */

                      /* If the DAQ-Liste only supports one fixed ECU
                         sampling rate, it can be declared below
                         to achieve compatibility with the ASAP2 standard.
                         Otherwise description of the possible ECU
                         sampling rates in QP_BLOB      */
        int;          /* Period definition : Basic scaling unit in
                         CSE defined in ASAP1b (CSE=Code for Scaling Unit) */
        long;         /* Period definition : Rate in Scaling Units */
      };


      taggedstruct {

        "DISPLAY_IDENTIFIER" char[32];

        block "QP_BLOB" struct {

          uint;                    /* Number of the DAQ-List 0..n                */

          taggedstruct {
            "LENGTH" uint;     /* Length of the DAQ-Liste, maximum number of
                                  the useable ODTs                          */

            "CAN_ID_VARIABLE";
                               /* CAN-Message-ID is variable             */

            "CAN_ID_FIXED" ulong;
                               /* CAN-Message-ID of the DTOs is fixed,
                                  Default DTO
                                  Bit31 = 1: extended Identifier
                                  Bit31 = 0: standard Identifier             */

                               /* Not applied if the ECU uses the DTM-Id    */

            ("RASTER" uchar; )*;
                               /* Supported CCP Event Channel Names
                                  of this DAQ List */

            ("EXCLUSIVE" int; )*;
                               /* Exclusion of other DAQ-Lists              */

            "REDUCTION_ALLOWED";
                               /* Data reduction possible                   */

            "FIRST_PID" uchar;     /* First Packet ID (PID) of the DAQ List */
          };
        };
      };
    } )*;


    /* Description of the available ECU Sampling Rates (Event Channels)      */
    (block "RASTER" struct {

       char [101];   /* CCP Event Channel Name                               */
       char [9];     /* Short Display Name of the Event Channel Name         */
       uchar;        /* Event Channel No., used for CCP START_STOP)          */
       int;          /* Period definition :  basic scaling unit in CSE
                        as defined in ASAP1b                                 */
       long;         /* ECU sample rate of the event channel,
                        period definition based on the basic scaling unit    */

       taggedstruct {
         ("EXCLUSIVE" uchar; )*;
                     /* Exclusion of other CCP Event Channels                */
       };

    }; )*;

    /* Group several event channels to form one combined event */
    /* e.g. group all cylinder synchronous events to one combined element */
    (block "EVENT_GROUP" struct {

       char [101];   /* Event group name */
       char [9];     /* Short name for the event group */
       taggedstruct {
         ("RASTER" uchar; )*;
       };
                     /* all event channels beloging to group
                        (CCP Event Channel Numbers for START_STOP)               */
    } )*;


    /* Description of the authentification process */
    block "SEED_KEY" struct {
       char[256];    /* Name of the Seed&Key DLL for CAL Priviledge,
                        including file-Extension without path */
       char[256];    /* Name of the Seed&Key DLL for DAQ Priviledge,
                        including file-Extension without path */
       char[256];    /* Name of the Seed&Key DLL for PGM Priviledge,
                        including file-Extension without path */
    };


/* Description of the checksum calculation process */
    block "CHECKSUM" struct {
       char[256];    /* Name of the Checksum DLL representing the ECU Algorithm,
                        including file-Extension without path */
};


block "TP_BLOB" struct {

      uint;    /* CCP Version,       High Byte: Version
                                    Low Byte : subversion (dec.)            */
      uint;    /* Blob-Version,      High Byte: Version
                                    Low Byte : subversion (dec.)            */
      ulong;   /* CAN-Message ID for 'Transmitting to ECU (CRM)'
```

```
                              Bit31 = 1: extended Identifier
                              Bit31 = 0: standard Identifier          */
ulong;   /* CAN-Message ID for 'Receiving from ECU (DTM)'
                              Bit31 = 1: extended Identifier
                              Bit31 = 0: standard Identifier          */
uint;    /* Logical CCP-Address of the (station address)           */
uint;    /* Byte order of Multiple-byte-items
                              1 = high Byte first, 2 = low byte first */
taggedstruct {

  block "CAN_PARAM" struct {
    uint;                  /* Quartz freq. of the elec. control unit  */
    uchar;                 /* BTR0                                    */
    uchar;                 /* BTR1                                    */
  };

  "BAUDRATE" ulong;          /* Baud rate in Hz.                           */
  "SAMPLE_POINT" uchar;    /* sampling point of time in percent       */
  "SAMPLE_RATE" uchar;     /* number of samples per Bit (1 oder 3)    */
  "BTL_CYCLES" uchar;      /* number of BTL-cycles                    */
  "SJW" uchar;             /* SJW-parameter in BTL-cycles             */
  "SYNC_EDGE" enum {
     "SINGLE" = 0,         /* Synchronisation only on fallende edge    */
     "DUAL" = 1            /* Synchr. on falling and rising edge      */
  };

  "DAQ_MODE" enum {        /* mode of cylcic data acquisition         */
     "ALTERNATING" = 0,    /* ECU is sending one ODT per cycle         */
     "BURST" = 1           /* ECU is sending a complete DAQ            */
  };

  "BYTES_ONLY";            /* ECU supports max. elements of one Byte size */

  "RESUME_SUPPORTED";      /* ECU supports the Resume function       */
  "STORE_SUPPORTED";       /* ECU supports the Store function        */

  "CONSISTENCY" enum {
     "DAQ" = 0,            /* consistency of a complete DAQ ist guaranteed */
     "ODT" = 1             /* consistency of a complete ODT ist guaranteed */
  };

  "ADDRESS_EXTENSION" enum {  /* address extension                        */
     "DAQ" = 0,               /* ECU supports only one Address extension
                                 within an DAQ                      */
     "ODT" = 1                /* ECU supports only one Address extension
                                 within an ODT                      */
  };

  block "CHECKSUM_PARAM" struct {
    uint;                /* checksum calculation procedure
                         standard types not yet defined,
                         if greater of equal 1000 : manufacturer specific  */

    ulong;               /* Maximum block length used by an ASAP1a-CCP
                            command, for checksum calculation procedure  */

    taggedstruct {
      "CHECKSUM_CALCULATION" enum {
         "ACTIVE_PAGE" = 0,
         "BIT_OR_WITH_OPT_PAGE" = 1
      };
    };
  };

  (block "DEFINED_PAGES" struct {
     struct {
        uint;            /* Logical No. of the memory page (1,2,..)   */
        char[101];       /* Name of the memory page                   */
        uint;            /* Adress-Extension of the memory page (only
                            Low Byte significant)                      */
        ulong;           /* Base address of the memory page           */
        ulong;           /* Length of the memory page in Bytes         */
     };
     taggedstruct {
       "RAM";              /* memory page in RAM */
       "ROM";              /* memory page in ROM */
       "FLASH";            /* memory page in FLASH */
       "EEPROM";           /* memory page in EEPROM */
       "RAM_INIT_BY_ECU";  /* memory page is initialised by ECU start-up */
       "RAM_INIT_BY_TOOL"; /* RAM- memory page is initialised by the MCD
                              system */
```

```
                "AUTO_FLASH_BACK";  /* RAM memory page is automatically flashed back */
                "FLASH_BACK";       /* feature available to flash back the RAM memory page */
                "DEFAULT";          /* memory page is standard (fallback mode) */
              };
           } )*;

           ( "OPTIONAL_CMD"  uint; )*; /* CCP-Code of the optional command available
                                          in the ECU. It is recommended to declare all
                                          non-standard ECU commands here    */

         };

       };

   /* for CHARACTERISTIC and AXIS_PTS and MEMORY_LAYOUT */
       "DP_BLOB" struct {
         uint;  /* Address extension of the calibration data
                        (only Low Byte significant) */
         ulong; /* Base address of the calibration data */
         ulong; /* Number of Bytes belonging to the calibration data  */
       };

   /* for MEASUREMENT */
       "KP_BLOB" struct {
         uint;  /* Address extension of the online data
                        (only Low Byte significant) */
         ulong; /* Base address of the online data   */
         ulong; /* Number of Bytes belonging to the online data (1,2 or 4) */
         taggedstruct {
            ("RASTER" uchar; )*;
                     /* Array of event channel initialization values */
          };
       };
};
```

8.7 Win32 API for the ASAP1a CCP Seed&Key Algorithm DLL

Author : Michael Rossmann, SIEMENS

In order to have a common implementation of the Seed&Key algorithms used for getting access to a locked, CCP accessed ECU, the following API is proposed:

Function name:	ASAP1A_CCP_ComputeKeyFromSeed
Parameter 1:	Pointer to the Seed data, retrieved from ECU’s GET_SEED cmd, 4 BYTEs of data.
Parameter 2:	Pointer to 6 BYTEs of data, returning the calculated Key.

An example of an implementation of such an API written in C++ is like follows:

```
/*
// Header file for ASAP1a CCP V2.1 Seed&Key Algorithm
*/

#ifndef _SEEDKEY_H_
#define _SEEDKEY_H_

#ifndef DllImport
#define DllImport    __declspec( dllimport )
#endif
#ifndef DllExport
#define DllExport    __declspec( dllexport )
#endif
#ifdef SEEDKEYAPI_IMPL  // only defined by implementor of SeedKeyApi
#define SEEDKEYAPI DllExport __cdecl
#else
#define SEEDKEYAPI DllImport __cdecl
#endif

#ifdef __cplusplus
extern "C" {
#endif

BOOL SEEDKEYAPI ASAP1A_CCP_ComputeKeyFromSeed (BYTE *Seed,
                                               unsigned short SizeSeed,
                                               BYTE *Key,
                                               unsigned short MaxSizeKey,
                                               unsigned short *SizeKey);

// Seed:    Pointer to seed data
// SizeSeed:Size of seed data (length of ‚Seed‘)
// Key:     Pointer, where DLL should insert the calculated key data.
// MaxSizeKey: Maximum size of ‚Key‘.
// SizeKey: Should be set from DLL corresponding to the number of data
//          inserted to ‚Key‘ (at most ‚MaxSizeKey‘)
// Result:  The value FALSE (= 0) indicates that the key could not be
//          calculated from seed data (e.g. ‚MaxSizeKey‘ is too small).
//          TRUE (!= 0) indicates success of key calculation.

#ifdef __cplusplus
}
#endif
#endif //_SEEDKEY_H_
```

```
//
// Implementation of Seed&Key DLL for ASAP1a CCP V2.1
//

#include <windows.h>
#include <memory.h>
#define SEEDKEYAPI_IMPL
#include "..\seedkey.h"

extern "C" {

BOOL SEEDKEYAPI ASAP1A_CCP_ComputeKeyFromSeed (BYTE *Seed,
                                               unsigned short SizeSeed,
                                               BYTE *Key,
                                               unsigned short MaxSizeKey,
                                               unsigned short *SizeKey )
{
      /* ... implementation of seed&key algorithm.
            The result should be written in Key.
            the result of the CCP 2.1 cmd GET_SEED.
      */
      MessageBox(NULL, "Call to ASAP1A_CCP_ComputeKeyFromSeed", "Seed&Key",
MB_OK);
}

}
```

8.8 Win32 API for the ASAP1a Checksum Algorithm DLL

Author : Michael Rossmann, SIEMENS

In order to have a common interface to the implementation of the checksum algorithms used for verifying ECU calibration and program data, the following API is proposed:

Function name:	BOOL CalcChecksum(struct TRange *ptr, int nRanges, BYTE *pnChecksum, int *pnSignificant, WORD nFlags)
Parameter 1:	Pointer to an array of ranges, stored in structures of type TRange.
Parameter 2:	Number of ranges stored in the array which parameter 1 points to.
Parameter 3:	Pointer to a byte array where the checksum has to be stored. A maximum of 8 bytes will be written by the DLL, so the caller should reserve space for 8 bytes of data.
Parameter 4:	Length of actually calculated checksum (1 ... 8)
Parameter 5:	Flag field for commanding the way how the algorithm should work. Currently, only bit 0 is defined: Bit 0 = 0 means: pnChecksum shall receive the result of the checksum calculation of the algorithm. Bit 0 = 1 means: pnChecksum points to a checksum which shall be compared within the DLL with the checksum calculated by the algorithm. Return TRUE if checksums are identical, FALSE otherwise. All other bits are reserved and should be set to zero.

A TRange is defined as follows:

```
struct TRange
{
      char              *pMem;
      unsigned long     lLen;
}
```

The Calling convention is like defined in „WIN32 API specification for ASAP1b“, chapter 2.4.

An example of an implementation of such an API written in C++ is like follows:

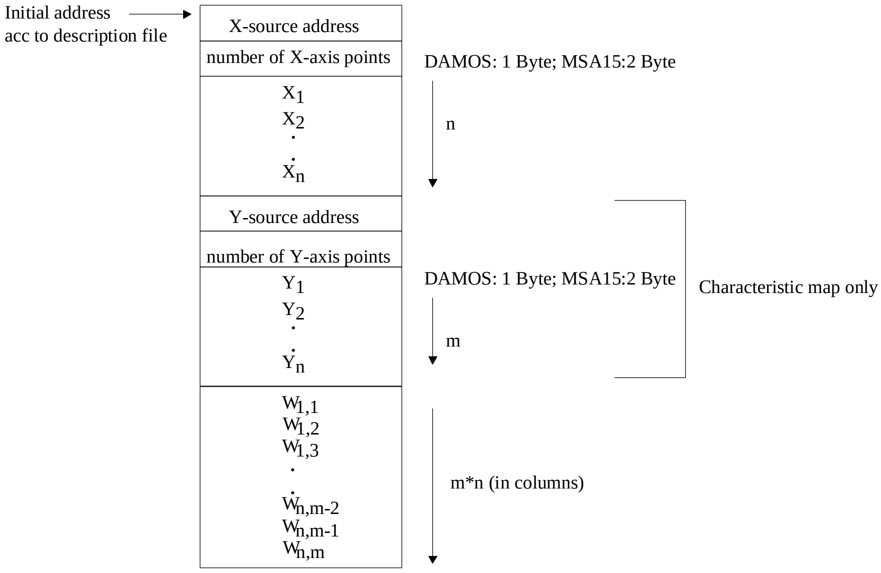
```
/*
// checksum.h
// Header file for ASAP1a CCP V2.1 Checksum Algorithm
*/
#ifndef _CHECKSUM_H
#define _CHECKSUM_H

#ifdef __cplusplus
extern "C" {
#endif

#ifndef DllImport
#define DllImport    __declspec( dllimport )
#endif
#ifndef DllExport
#define DllExport    __declspec( dllexport )
#endif

#ifdef CHECKSUMAPI_IMPL // only defined by implementor of ChecksumApi
#define CHECKSUMAPI DllExport __cdecl
#else
#define CHECKSUMAPI DllImport __cdecl
#endif

struct TRange
{
      char              *pMem;
      unsigned long     lLen;
};

#ifdef __cplusplus
extern "C" {
#endif

BOOL CHECKSUMAPI CalcChecksum(struct TRange *ptr, int nRanges, BYTE
*pnChecksum, int *pnSignificant, WORD nFlags);

#ifdef __cplusplus
}
#endif

#endif //_CHECKSUM_H
```

```
//
// Implementation of checksum DLL for ASAP1a CCP V2.1
//

#include <windows.h>
#include <memory.h>
#define CHECKSUMAPI_IMPL
#include "checksum.h"

extern "C" {

BOOL CHECKSUMAPI CalcChecksum(struct TRange *ptr, int nRanges,
                              BYTE *pnChecksum, int *pnSignificant,
                              WORD nFlags)
{
    int i;
    unsigned long crc32_value = 0xffffffff;   /* pre-condition - bits all
                                                 ones */
    BOOL bRet = TRUE;

    *pnSignificant = 4;

    for(i = 0; i < nRanges; i++)
    {
        crc32_value = calculate_crc32(ptr[i].pMem, ptr[i].lLen,
                                      crc32_value);
    }
    crc32_value = crc32_value ^ 0xffffffff;  /* post-condition - one's
                                                complement */

    if(nFlags & 0x0001)
    {
        if(memcmp(&crc32_value,pnChecksum,*pnSignificant) == 0)
            bRet = TRUE;
        else
            bRet = FALSE;
    }
    memcpy(pnChecksum, &crc32_value, sizeof(unsigned long));

    return bRet;
}

} // extern "C"
```

9 Appendix A: Record layouts

BOSCH: DAMOS, MSA15

CC, CD, AP distribution



FV, FVB, FCC, FCD, GCC, GCD, ASCII:

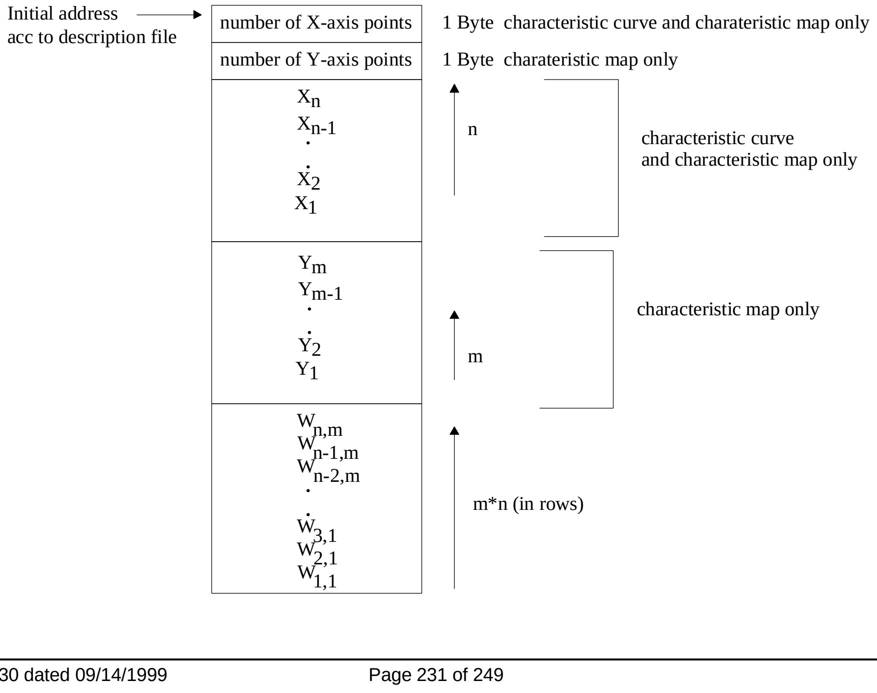
Address points directly to the function value

VFV, VFCC, VFCD:

Function values are addressed indirectly (via vector table)

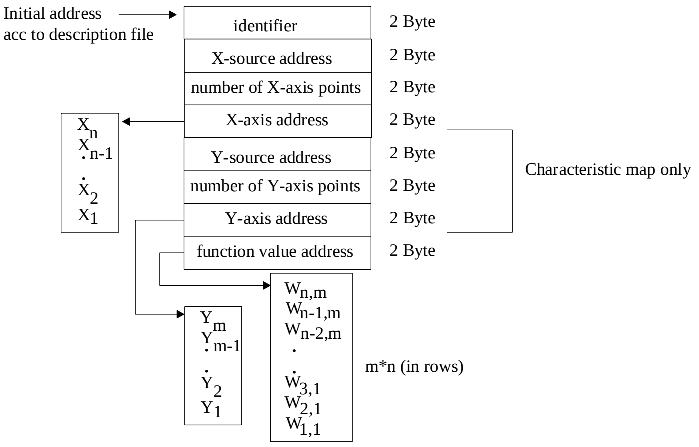
BOSCH: KEBUSS

FV, CC, CD

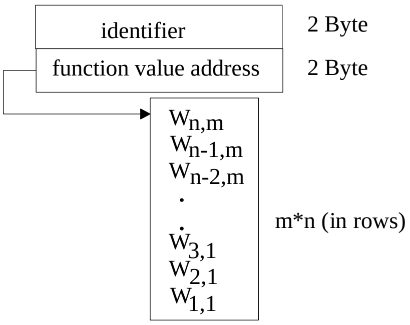


BOSCH: C-DAMOS

CC, CD, AP distribution



FCC, FCD, GCC, GCD, ASCII



FV, FVB:

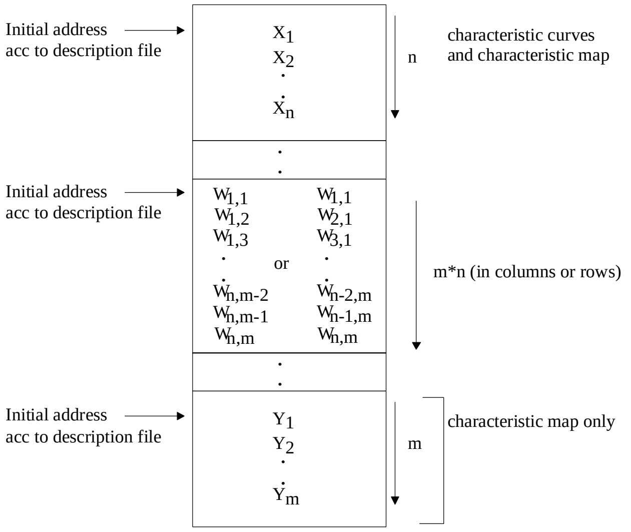
Address points directly to the value(s)

VFCC, VFCD:

Indirect addressing via vector table (in all other respects identical to FCC, FCD)

Siemens: Record layout

FV, CC, CD (other types???)



10 Appendix B: IEEE-Floating-Point-Format

Sign	Biases Exponent					Significant						
S	e7	e6	...	e1	e0	b1	b2	b3	...	b21	b22	b23

31 23 0

Table 6: IEEE-Floating-Point-Format (32-Bit)

Representation of real numbers: $(-1)^s * 2^E * b_{0\Delta} b_1 b_2 b_3 ... b_{23}$

s: 0 or 1
E: any integer between -126 and +127 (E = e - 127)
b_i: 0 or 1 (where b_0 = 1)

$$\text{RealNumber} = (-1)^s \quad * \quad 2^{(-127)+\sum_{i=0}^{7}\left(e_i * 2^i\right)} \quad * \quad \sum_{i=0}^{23}\left(\frac{b_i}{2^i}\right)$$

where b_0 = 1

Sign	Biases Exponent					Significant						
S	e10	e9	...	e1	e0	b1	b2	b3	...	b49	b50	b51

62 51 0

Table 7: IEEE-Floating-Point-Format (64-Bit)

11 Appendix C: modifications since version 1.0, and version 1.21

12 Appendix D: modifications since version 1.21

13 Appendix E: Glossary

ABUS

Automobile Bit-serial Universal Interface

Record layout

Description of the data structure with which an adjustable object of the control unit program is stored in memory.

ADC

Analog-Digital Converter

ADEX

Application Data Exchange System: Communication between a higher-order application system and the control units of a vehicle for the exchange of data.

AS

Application System

ASAP

Arbeitskreis zur Standardisierung von Applikationshilfsmitteln (Working Group on the Standardisation of Application Tools)

ASAP 1b

Programmed standardised interface in the application system for access to the ASAP device (see Figure 10).

ASAP 2

Standardised interface for the description data (description of control unit program: see Figure 10 and Figure 14).

ASAP device

The concept 'ASAP device' denotes a driver in the application system and the corresponding application device and control unit (if a control can be assigned).

ASAP2 metalanguage

Formal description language for the description of non-standardised, interface-specific ASAP2 description data.

ASCII

Adjustable object of the string type.

AuSy

Automation System

Description data

For the application of a control unit program it must be possible to display and edit adjustable objects. In addition, it must be possible to display, collect and store measurements. This requires a description of the control unit program, which must contain all information needed to read and write adjustable objects in the emulation memory and to collect measurements. Moreover, information is needed which describes the display format of the adjustable and measurement objects.

Byte order

This concept denotes how the inidvidual bytes of a multibyte of the control unit program are to be interpreted (Intel format or Motorola format).

CAN

Control Area Network

C-DAMOS deposit

This concept denotes a specific data structure with which the adjustable objects (characteristics) are deposited in memory (BOSCH control units).

D-bus

Diagnostics bus

DAMOS deposit

This concept denotes a specific data structure with which the adjustable objects (characteristics) are deposited in memory (BOSCH control units).

Display table

Method for the output of control unit internal measurements (BOSCH):

1) The application system manipulates an address table in the data area of the control unit program.

2) The control unit program reads these tables in a predefined time pattern and outputs the corresponding data on defined addresses in the dual-ported RAM.

EPROM identifier

String in the data area of the control unit program for the description of the control unit program.

Fixed characteristic curve, fixed characteristic map

Characteristic curve or characteristic map in which the axis point values are contained as absolute or difference values in the data record but are calculated as follows (equidistant axis points):

$$Apo_i = offset + (i - 1)*2^{shift} \qquad i = \{ 1...numberofaxispoints\}$$

Both parameters <offset> and <shift> are contained either in the description file or in the data record of the control unit program.

Function orientation

For the structuring of projects involving a very large number of adjustable objects and measurement objects, functions can be defined in ASAP2. These functions shall be used in the application system to allow the selection lists for the selection of the adjustable objects and measuring channels to be represented in a structured manner on the basis of functional viewpoints.

Group characteristic curve, group characteristic map

In a number of BOSCH control unit programs, "group characteristic curve" or "group characteristic map" denotes those characteristic curves or characteristic maps that have axis point distributions in common with other characteristic curves or characteristic map. Such an axis point distribution is allocated not to a single characteristic curve or characteristic map but to several characteristic curves and characteristic maps. If such an axis point distribution is changed, the behaviour of all allocated characteristic curve or characteristic map changes accordingly.

HW interface

Hardware interface, interface converter

KEBUSS deposit

This concept denotes a specific data structure with which the adjustable objects (characteristics) are deposited in memory (BOSCH control units).

Characteristic block

List of characteristics of the same data type (equal conversion method), which are stored sequentially in the data area of the control unit program (array) and which are considered as representing an adjustable object.

MCD

Measuring, Calibration and Diagnostics system

MSA15 deposit

This concept denotes a specific data structure with which the adjustable objects (characteristics) are deposited in memory (BOSCH control units).

MT

Module Type

ROM

Read-Only Memory

SG

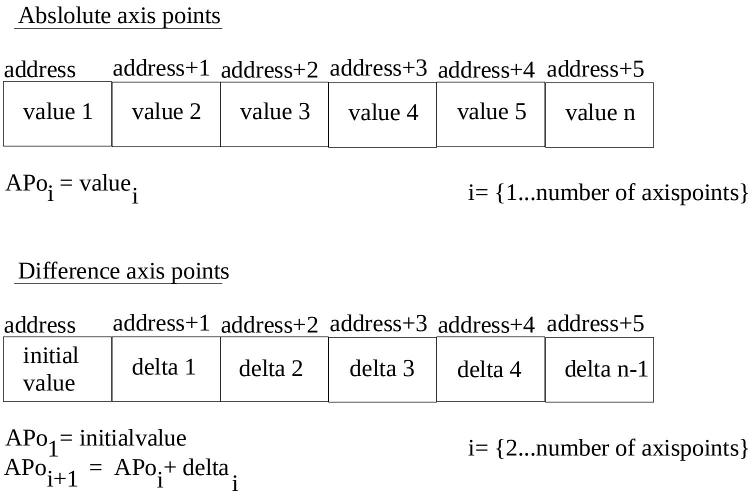
Steuergerät (control unit)

SIEMENS deposit

This concept denotes a specific data structure with which the adjustable objects (characteristics) are deposited in memory (SIEMENS control units).

Deposit of axis points

This concept describes how the axis point values of a characteristic curve or characteristic map are deposited in memory:



Abslolute axis points

address	address+1	address+2	address+3	address+4	address+5
value 1	value 2	value 3	value 4	value 5	value n

APo_i = $value_i$ i= {1...number of axispoints}

Difference axis points

address	address+1	address+2	address+3	address+4	address+5
initial value	delta 1	delta 2	delta 3	delta 4	delta n-1

APo_1= initialvalue i= {2...number of axispoints}

APo_{i+1} = APo_i+ $delta_i$

Figure 17: data deposition

Verbal conversion table

Conversion table for the visualisation of bit patterns. This conversion method is used for special measurements. As a rule, parts of the measurements are masked out via bit masks. Each bit sample of the quantity thus obtained is allocated a string in the verbal conversion table, which describes the state of this quantity.

14 Appendix F: ASAP2 keywords

15 Index