Format Specification
MDF Format

Version 3.3.1

Document Management of Specification

Revision History			
Document Version	**Date**	**Editor**	**Description**
1.0	2006-09-25	Osr	Conversion of old document and upgrade to MDF version 3.1: Description of missing blocks (CE, CD) and missing conversion formulas.
1.1	2006-10-31	Osr	Example for start offset, changes and additions for different elements.
1.2	2006-11-02	Osr	Added MDF revision history, changed layout
1.3	2006-11-09	Osr	Added TR block
1.4	2008-04-09	Osr	Extension to MDF 3.2: - exact time stamp for HDBLOCK - changed LINK from signed to unsigned integer
1.5	2008-09-23	Osr	Fixed error in HBLOCK Added example for start time in ns / UTC offset
1.6	2009-01-07	Osr	Extension to MDF 3.3 - added SRBLOCK
1.7	2009-04-21	Osr	Added code page for IDBLOCK Improved description of SRBLOCK Removed support of 10 Byte double (extended precision) channel data type because not specified by IEEE 754. Added remark on time stamps relative to HDBLOCK
1.8	2013-05-07	Osr	Fixed bug for number of dimensions of CDBLOCK
1.9	2013-06-25	Osr	Added hint that last member of CGBLOCK is only valid for MDF 3.30 or higher.
1.10	2014-03-11	Osr	Added description of unfinalized MDF (IDBLOCK)

Table of Contents

1 Introduction 5

1.1 Outline 5

1.2 Abbreviations 5

2 MDF Versions 6

2.1 Version History 6

2.2 Version Handling 6

2.3 Rules to Ensure MDF Compatibility between Versions 7

3 MDF General Block Format 7

3.1 Definition of Data Types Used 9

3.2 Overview of Block Types Used 9

3.3 The File Identification Block IDBLOCK 10
3.3.1 Block structure of IDBLOCK 10

3.4 The Header Block HDBLOCK 13
3.4.1 Block structure of HDBLOCK 13

3.5 The Text Block TXBLOCK 14
3.5.1 Block structure of TXBLOCK 14

3.6 The Program-Specific Block PRBLOCK 14
3.6.1 Block structure of PRBLOCK 14

3.7 The Trigger Block TRBLOCK 14
3.7.1 Block structure of TRBLOCK 14

3.8 The Sample Reduction Block SRBLOCK 15
3.8.1 Block structure of SRBLOCK 15
3.8.2 Motivation for Sample Reduction 15
3.8.3 Layout of Reduced Samples 16

3.9 The Data Group Block DGBLOCK 17
3.9.1 Block structure of DGBLOCK 17

3.10 The Channel Group Block CGBLOCK 17
3.10.1 Block structure of CGBLOCK 17

3.11 The Channel Block CNBLOCK 18
3.11.1 Block structure of CNBLOCK 18

3.12 The Channel Conversion Block CCBLOCK 20
3.12.1 Block structure of CCBLOCK 20
3.12.2 CCBLOCK – Linear Function with 2 Parameters 20
3.12.3 CCBLOCK – Polynomial Function with 6 Parameters 21
3.12.4 CCBLOCK – Tabular, with Interpolation 21
3.12.5 CCBLOCK – Tabular 21
3.12.6 CCBLOCK – Exponential 21
3.12.7 CCBLOCK – Logarithmic 22
3.12.8 CCBLOCK –Rational Conversion Formula 22
3.12.9 CCBLOCK – ASAM-MCD2 Text formula 23
3.12.10 CCBLOCK – ASAM-MCD2 Text Table 23
3.12.11 CCBLOCK – ASAM-MCD2 Text Range Table 23
3.12.12 CCBLOCK – Data Structure Date 24
3.12.13 CCBLOCK – Data Structure Time 24

3.13 The Channel Dependency Block CDBLOCK 25

3.13.1 Block structure of CDBLOCK 25

3.14 The Channel Extension Block CEBLOCK 26
3.14.1 Block structure of CEBLOCK 26
3.14.2 DIM block supplement 26
3.14.3 Vector CAN block supplement 26

4 MDF Data Format 27

4.1 Data Format for Sorted MDF Files 27

4.2 Data Format for Unsorted MDF Files 28

4.3 Reading Signal Values from a Data Record 28
4.3.1 Example 1: Little Endian (Intel) Byte Order 29
4.3.2 Example 2: Big Endian (Motorola) Byte Order 30

1 Introduction

MDF (**M**easurement **D**ata **F**ormat) is a binary file format that can be used for recording, exchanging, and post-measurement analysis of measurement data. MDF is well established in the automotive industry. The MDF format provides

- exchange between a number of tools in the automotive industry
- compact description of data
- fast access to general file info independent of the file length

This document serves to specify MDF version 3.3.

1.1 Outline

Chapter 2 provides a history of the MDF format and describes the conventions for extending and updating the format. Chapter 3 specifies the structure of the MDF file and the various block types. Chapter 4 describes the structure of the data block and shows how to read signal values.

1.2 Abbreviations

ASAM	Association for Standardization of Automation and Measuring Systems
CCBLOCK	Channel Conversion BLOCK
CDBLOCK	Channel Dependency BLOCK
CEBLOCK	Channel Extension BLOCK
CGBLOCK	Channel Group BLOCK
CNBLOCK	Channel BLOCK
DGBLOCK	Data Group BLOCK
HDBLOCK	Header BLOCK
IDBLOCK	Identification BLOCK
LSB	Least Significant Bit
MDF	Measure Data Format
MSB	Most Significant Bit
NIL	NIL pointer (0x00000000)
PRBLOCK	Program BLOCK
SRBLOCK	Sample Reduction Block
TRBLOCK	Trigger Block
TXBLOCK	Text BLOCK
UTC	Universal Coordinated Time

2 MDF Versions

The MDF format has been extended several times since its creation in 1991. This chapter provides a brief history of the major changes to the MDF format and instructions how to handle future updates.

2.1 Version History

MDF was developed in 1991 by Vector Informatik GmbH in co-operation with Robert Bosch GmbH. Since then, the format was only extended in smaller details, for instance to support new data types for strings, fields for ASAM compatible signal names, or new conversion rules. In the meantime, the MDF format has become a standard for measurement data in the automotive field.

Major Revisions:

Year	Version	Description
1991	2.03	First official release of MDF.
1996	2.11	New conversion type (CCBLOCK): ASAM-MCD2 text.
2000	2.12	New conversion type (CCBLOCK): ASAM-MCD2 text table. New field (CNBLOCK): "Long Signal Name".
2000	2.13	New extension type (CEBLOCK): Vector CAN block.
2001	2.15	New data types (CNBLOCK): "String" and "Byte Array". New conversion types (CCBLOCK): "Date" and "Time".
2002	3.00	New conversion type (CCBLOCK): ASAM-MCD2 text range table. New fields (CNBLOCK): "Display Name" and "Additional Byte Offset".
2005	3.01	N-dimensional dependency in CDBLOCK.
2006	3.10	New signal data types (CNBLOCK) to define a specific Byte order for integer and floating point signal values (can differ from default Byte order). Allow crossing of Byte boundaries for Integer signals with bit offset > 0.
2008	3.20	Exact start time and time quality information (HDBLOCK) LINK changed from signed to unsigned integer (maximum file size of a MDF file thus extended from (approx.) 2 GB to 4 GB)
2009	3.30	Sample reduction for a channel group (SRBLOCK) Specification of code page for strings (IDBLOCK) channel data type 10 Byte double (extended precision) obsolete.
2014	3.31	Identification and flags for "unfinalized" MDF in IDBLOCK (version independent feature)

2.2 Version Handling

The MDF version number consists of three digits, a major version number, a minor version number and a revision number. Normally only the major and minor version numbers are stated, e.g. V3.2 with major version number "3" and minor version number "2". In the IDBLOCK, an additional revision number is appended. For the current version, for example, the version string reads "3.20" which must be interpreted as V3.2, revision "0" (and not as minor version number "20"!). The restriction to three digits implies that both minor version number and revision number must never exceed nine.

Each change in the MDF file format must result in a higher version number, i.e. the three-digit-number consisting of major, minor and revision number must be larger than that of the previous version (e.g. 310 > 301 for versions 3.10 and 3.01).

Here are some general guidelines when to increase which of the numbers:

Element	Description
major version number	Changes in the MDF file format that may result in a misinterpretation of the data when evaluated by some older tool require a change of the major version number. A tool that evaluates a MDF file that has a higher major version number than what is supported by the tool should reject reading the file and generate a warning or error message.

minor version number	Changes in the MDF file format that may not result in a misinterpretation of the data when evaluated by some older tool require only a change of the minor version number. Still the older tool may miss new features/information stored in the new MDF version. Normally it would simply ignore them and generate a warning message.
revision number	Changes in the MDF file format that do not affect the interpretation of an older tool may be done by changing the revision number only.

2.3 Rules to Ensure MDF Compatibility between Versions

To ensure a high degree of compatibility, new entries are only appended to the blocks. This implies the following consequences for the evaluation tool:

1. If a tool detects a block that has a smaller block length than expected by the version supported by the tool, the additional fields are set to their default values. This is the case, if the tool supports a later MDF version than the file is encoded in.

2. If the tool detects a block that has a larger block length than expected by the version supported by the tool, the additional fields are ignored. This is the case, if the tool supports a previous MDF version compared to the one the file is encoded with. In this case, the tool may interpret the measurement data in the file incorrectly; thus the MDF major version number has to be increased when introducing such a change.

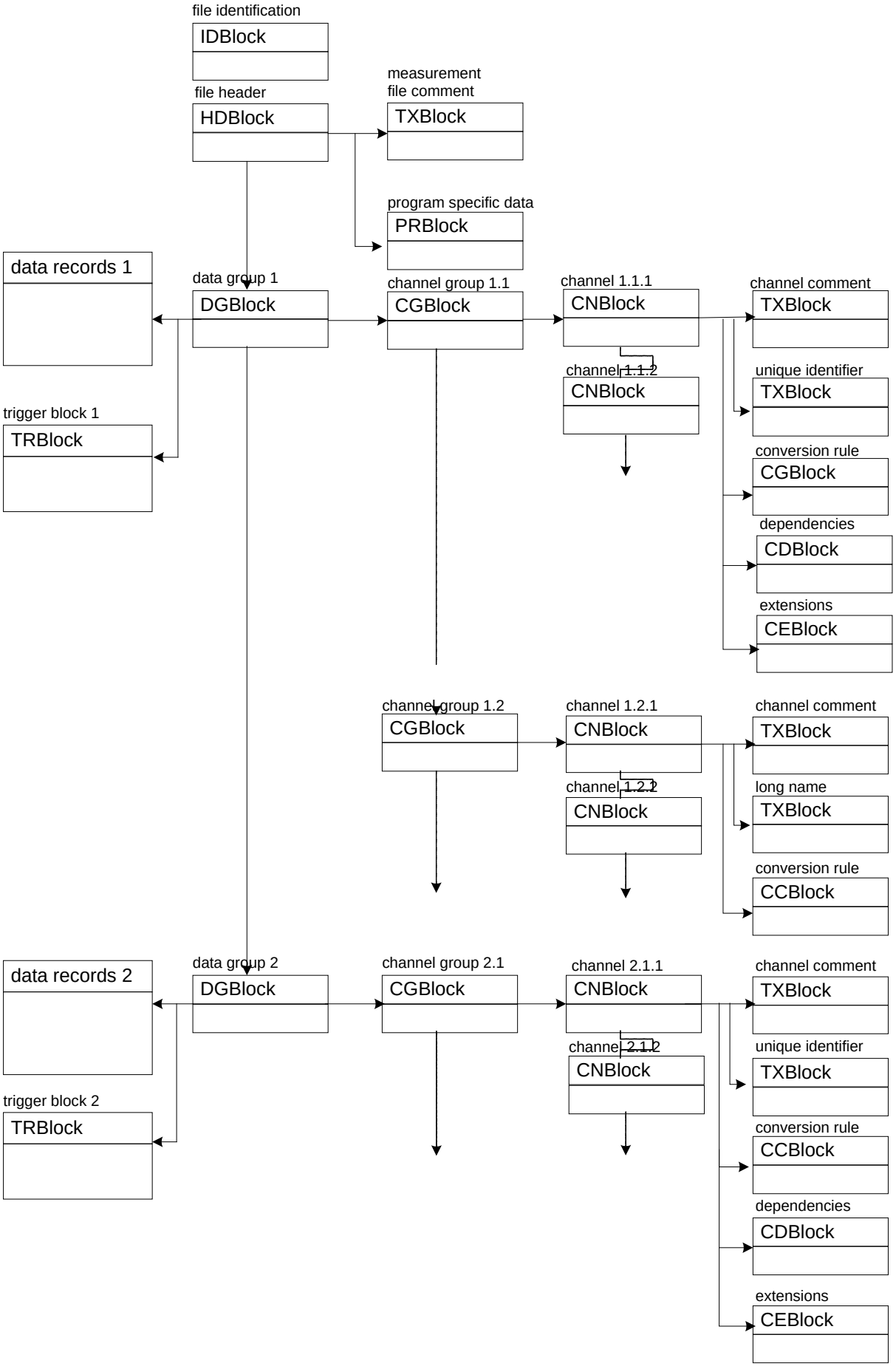
3 MDF General Block Format

The MDF file is composed of a series of blocks. Each block consists of a number of contiguous Bytes and can be seen as a record or structure of data fields. There are different types of blocks. Blocks can include pointers to other blocks that are stored in a data field of type LINK. A link is an absolute Byte position within the file, starting at the beginning of the file. Thus a normally tree-like hierarchy of blocks is formed.

The file always starts with the file identifier block IDBLOCK. The file header block HDBLOCK follows at byte position 64. All other blocks are linked via LINKs and can be stored in arbitrary order in the file.

With the exception of the IDBLOCK and the data block, the "Block Size" field indicates the size of each block. To ensure upward compatibility, the block sizes specified in this document should be regarded as the minimum block sizes in each case.

The following figure gives an overview of the general block structure of an MDF file:


file identification
IDBlock
file header
HDBlock
measurement
file comment
TXBlock
program specific data
PRBlock
data records 1
data group 1
DGBlock
channel group 1.1
CGBlock
channel 1.1.1
CNBlock
channel comment
TXBlock
channel 1.1.2
CNBlock
unique identifier
TXBlock
trigger block 1
TRBlock
conversion rule
CGBlock
dependencies
CDBlock
extensions
CEBlock
channel group 1.2
CGBlock
channel 1.2.1
CNBlock
channel comment
TXBlock
channel 1.2.2
CNBlock
long name
TXBlock
conversion rule
CCBlock
data records 2
data group 2
DGBlock
channel group 2.1
CGBlock
channel 2.1.1
CNBlock
channel comment
TXBlock
channel 2.1.2
CNBlock
unique identifier
TXBlock
trigger block 2
TRBlock
conversion rule
CCBlock
dependencies
CDBlock
extensions
CEBlock

3.1 Definition of Data Types Used

Data Type	Format
CHAR	1 byte representing a character (C data type: char). The storage of character strings may or may not be terminated by a zero byte.
UINT8	8-bit unsigned integer
UINT16	16-bit unsigned integer
INT16	16-bit signed integer
UINT32	32-bit unsigned integer
UINT64	64-bit unsigned integer
BOOL	Boolean variable, stored as 16-bit value If contents != 0 then TRUE, if contents == 0 then FALSE
REAL	Floating-point compliant with IEEE 754, double precision (64 bits)
LINK	32-bit unsigned integer, used as byte position within the file. If a LINK is nil (corresponds to 0), this means the corresponding block does not exist and the LINK cannot be de-referenced. Remember that all blocks except the IDBLOCK and HDBLOCK are optional.

3.2 Overview of Block Types Used

Block Type	Description	Purpose
IDBLOCK	Identification block	Identification of the file as MDF file and MDF version
HDBLOCK	Header block	General description of the measurement file
TXBLOCK	Text block	Contains a string with a variable length
PRBLOCK	Program block	Contains proprietary data of the application generating the MDF file
DGBLOCK	Data group block	Description of data block that may refer to one or several channel groups
CGBLOCK	Channel group block	Description of a channel group, i.e. signals which are always measured jointly
CNBLOCK	Channel block	Description of a channel
CCBLOCK	Channel conversion block	Description of a conversion formula for a channel
TRBLOCK	Trigger block	Description of a trigger event
CDBLOCK	Channel dependency block	Description of dependency between channels
CEBLOCK	Channel Extension block	Additional information about the data source of the channel
SRBLOCK	Sample Reduction block	Description of a sample reduction, i.e. alternative data rows for a channel group with usually lower number of samples.

3.3 The File Identification Block IDBLOCK

The IDBLOCK always begins at file position 0 and has a constant length of 64 Bytes. It contains information to identify the file. This includes information about the source of the file and general format specifications.

3.3.1 Block structure of IDBLOCK

Data type	Count	Description
CHAR	8	**File identifier,** always contains "MDF␣␣␣␣␣" ("MDF" followed by five spaces, no zero termination), except for "unfinalized" MDF files. Unfinalized MDF files are marked by a file identifier "UnFinMF␣" ("UnFinMF" followed by one space, no zero termination). Details see 3.3.2 Unfinalized MDF.
CHAR	8	**Format identifier**, a textual representation of the format version for display, e.g. "3.30" (including zero termination) for version 3.3 revision 0 (see section 2 MDF Version), or "3.30␣␣␣␣" (followed by spaces, no zero termination required if 4 spaces).
CHAR	8	**Program identifier**, to identify the program which generated the MDF file
UINT16	1	**Default Byte order** used for this file (can be overruled for values of a signal in CNBLOCK) 0 = Little Endian (Intel order) Any other value = Big Endian (Motorola order)
UINT16	1	**Default floating-point format** used for this file (can be overruled for values of a signal in CNBLOCK) 0 = Floating-point format compliant with IEEE 754 standard 1 = Floating-point format compliant with G_Float (VAX architecture) (obsolete) 2 = Floating-point format compliant with D_Float (VAX architecture) (obsolete)
UINT16	1	**Version number** of MDF format, i.e. 330 for this version
UINT16	1	**Code Page number** The code page used for all strings in the MDF file except of strings in IDBLOCK and string signals (string encoded in a record). Value = 0: code page is not known. Value > 0: identification number of extended ASCII code page (includes all ANSI and OEM code pages) as specified by Microsoft, see http://msdn.microsoft.com/en-us/library/dd317756(VS.85).aspx. The code page number can be used to choose the correct character set for displaying special characters (usually ASCII code ≥ 128) if the writer of the file used a different code page than the reader. Reading tools might not support the display of strings stored with a different code page, or they might only support a selection of (common) code pages. *Valid since version 3.30. Default value: 0* *Note: the code page is only for documentation and not required information. It might be ignored by tools reading the file, especially if they support only a MDF version < 3.30.*
CHAR	2	Reserved
CHAR	26	Reserved
UINT16	1	**Standard Flags** for unfinalized MDF Bit combination of flags that indicate the steps required to finalize the MDF file. For a finalized MDF file, the value must be 0 (no flag set). See also 3.3.2 Unfinalized MDF and the description of the custom flags below. There is no specific order required for the finalization steps of the currently defined standard flags. However, custom finalization steps (see below) may require a specific order, also with respect to the standard finalization steps. The value contains the following bit flags (Bit 0 = LSB):

		Bit 0: Update of record counters for CGBLOCKs required The value for the number of records (i.e. number of samples) in a CGBLCOK may be wrong (zero or not up-to-date). This flag must only be set in case a successful restoration is possible, i.e. if any of the following restrictions for the data block of the parent DGBLOCK is fulfilled: • The data block is limited by the end of the file (EOF). • The data block is limited by the start of another MDF block within the block hierarchy of the MDF file. • The data block is limited by a value not used as record ID (only in case a record ID is used). • The data block is limited by a record that contains an invalid time stamp, e.g. a time stamp smaller than the previous time stamp within the same channel group. A correction of the record counters usually requires the iteration over all records, so this may also be done during sorting an unsorted data block (see 4.2).
		Bit 1: Update of reduced sample counters for SRBLOCKs required The value for the number of reduced samples in a SRBLOCKs may be wrong (zero or not up-to-date). This flag must only be set in case a successful restoration is possible, i.e. if any of the following restrictions for the data block of the SRBLOCK is fulfilled: • The data block is limited by the end of the file (EOF). • The data block is limited by the start of another MDF block within the block hierarchy of the MDF file. • The data block is limited by a record that contains an invalid time stamp for the interval start, e.g. a time stamp smaller than the previous time stamp. A correction of the reduced sample counters usually requires the iteration over all records.
UINT16	1	**Custom Flags** for unfinalized MDF Bit combination of flags that indicate custom steps required to finalize the MDF file. For a finalized MDF file, the value must be 0 (no flag set). See also 3.3.2 Unfinalized MDF. Custom flags should only be used to handle cases that are not covered by the (currently known) standard flags (see above). The meaning of the flags depends on the creator tool, i.e. the application that has written the MDF file (see id_prog). Finalization should only be done by the creator tool or a tool that is familiar with all custom finalization steps required for a file from this creator tool.

Please note that both the default Byte order and the default floating point format apply to the complete file, i.e. all values (integers, floats) have to be interpreted accordingly. The only exception is that signal values can be stored with a different format in the data block. In this case, the format specified by the signal data type of the CNBLOCK overrules the default format.

For example, the complete file might use Little Endian (Intel) as default Byte order; nevertheless a signal in the data block was stored in Big Endian (Motorola) Byte order. In this case when reading the signal value from a record in the data block, the Bytes have to be interpreted with Big Endian (Motorola) order, i.e. on an Intel system the Bytes have to be swapped.

3.3.2 Unfinalized MDF

There may be situations where a tool just writing an MDF file ("creator tool") may not be able to successfully finalize it. This can be due to a premature termination of the program, e.g. by power off or an application error, etc. Depending on the writing strategy of the creator tool, the MDF file then may be left in an "unfinalized" state. This state may violate some format rules of MDF, or it does not show or contain all recorded data.

In many cases, a recovery of the unfinalized MDF file, i.e. a finalization by post-processing, is possible. The creator tool itself may offer this capability. Alternatively, the recovery may be done by another tool or even "manually" using some kind of editor.

However, if a third tool not being aware of the unfinalized state performs some post-processing step with the unfinalized MDF file (e.g. sorting), the recovery may become impossible and some part or even all of the contained data may be destroyed. Hence a not yet finalized MDF file should be marked as "unfinalized" in a way that any post-processing tool either rejects it or possibly is able to recover the file by itself.

The marking of an unfinalized MDF file is independent of the used MDF version and simply consists of a different file identifier. Such a marked file should be rejected by already existing tools because they will not find the required MDF file identifier.

In addition to the file identifier for unfinalized MDF, the last four bytes of the IDBLOCK are used to hold bit flags that indicate how the file can be finalized. The flags are separated into standard flags and custom flags:

- The standard flags indicate common finalization steps which may be supported by any tool. It is up to each tool if it supports finalization or not. It may also support only some of the standard finalization steps.
- The custom flags can be used by the creator tool to document own steps specific to this tool. These steps are not described here and can possibly be executed only by the creator tool itself.

Since the finalization steps may have to be executed in a specific order, a tool must not finalize an unfinalized file (not even partially) unless it knows all standard and custom flags and the necessary order of the respective finalization steps.

Any tool that successfully finalizes the MDF file must reset the bit flags and replace the file identifier with the normal MDF file identifier ("MDF␣␣␣␣␣"). Note that there is no extra file extension defined for an unfinalized MDF file.

3.4 The Header Block HDBLOCK

The HDBLOCK always begins at file position 64. It contains general information about the contents of the measured data file.

3.4.1 Block structure of HDBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "HD"
UINT16	1	**Block size** of this block in bytes (entire HDBLOCK)
LINK	1	Pointer to the first **data group block** (DGBLOCK)
LINK	1	Pointer to the measurement **file comment** text (TXBLOCK) (NIL allowed)
LINK	1	Pointer to **program block** (PRBLOCK) (NIL allowed)
Uint16	1	**Number of data groups** (redundant information)
CHAR	10	**Date** at which the recording was started in "DD:MM:YYYY" format
CHAR	8	**Time** at which the recording was started in "HH:MM:SS" format for locally displayed time, i.e. considering the daylight saving time (DST) Note: all time stamps in a time channel are only relative to the start time of the measurement (see remark on time channel type in CNBLOCK)
CHAR	32	**Author**'s name
CHAR	32	Name of the **organization or department**
CHAR	32	**Project** name
CHAR	32	**Subject / Measurement object**, e.g. vehicle information
UINT64	1	**Time stamp** at which recording was started in nanoseconds. Elapsed time since 00:00:00 01.01.1970 (local time) (local time = UTC time + UTC time offset) Note: the local time does not contain a daylight saving time (DST) offset! V*alid since version 3.20. Default value: 0* *See remark below*
INT16	1	**UTC time offset** in hours (= GMT time zone) For example 1 means GMT+1 time zone = Central European Time (CET). The value must be in range [-12, 12], i.e. it can be negative! V*alid since version 3.20. Default value: 0 (= GMT time)*
UINT16	1	**Time quality class** 0 = local PC reference time (Default) 10 = external time source 16 = external absolute synchronized time *Valid since version 3.20. Default value: 0*
CHAR	32	**Timer identification** (time source), e.g. "Local PC Reference Time" or "GPS Reference Time". *Valid since version 3.20. Default value: empty string*

Note:
The UINT64 time stamp in nanoseconds introduced in version 3.20 specifies the start time more precisely than the textual Date and Time fields. However, for compatibility to older tools, the textual fields must be filled in correctly even if using the UINT64 time stamp. On the other hand, if the UINT64 time stamp is zero, then the textual Date and Time fields must be used.

Examples for Time and Time stamp / UTC time offset

Central Europe (GMT+1) 2008-01-25 at 16:20:07 (Central European Time CET)

Date string:	"25:01:2008"	Time stamp [ns]:	1201278007000000000
Time string:	"16:20:07"	UTC offset [h]:	1

Central Europe (GMT+1) 2008-09-03 at 12:22:53 (Central European Summer Time CEST)

Date string:	"03:09:2008"	Time stamp [ns]:	1220440973000000000
Time string:	"12:22:53"	UTC offset [h]:	1

3.5 The Text Block TXBLOCK

The TXBLOCK contains an optional comment for the measured data file, channel group or signal, or the long name of a signal. The text length results from the block size.

3.5.1 Block structure of TXBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "TX"
UINT16	1	**Block size** of this block in bytes (entire TXBLOCK)
CHAR	variable	**Text** (new line indicated by CR and LF; end of text indicated by 0)

Note:
A file may reserve spaces to allow modifying the comment within its reserved range without the need to rewrite the entire MDF file.

3.6 The Program-Specific Block PRBLOCK

The PRBLOCK contains non-standardized data to exchange between the acquisition program and the evaluation program.

3.6.1 Block structure of PRBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "PR"
UINT16	1	**Block size** of this block in bytes (entire PRBLOCK)
CHAR	variable	**Program-specific data**

3.7 The Trigger Block TRBLOCK

The TRBLOCK serves to describe trigger events that occurred during the recording. The trigger events are related to a data group. For each trigger event, the time when the trigger occurred is recorded. As additional information, the pre and post trigger times of the trigger event can be stated, i.e. the time periods recorded before and after the trigger occurred. The comment might be used to give information about the trigger source or condition.

3.7.1 Block structure of TRBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "TR"
UINT16	1	**Block size** of this block in bytes (entire TRBLOCK)
LINK	1	Pointer to **trigger comment** text (TXBLOCK) (NIL allowed)
UINT16	1	**Number of trigger events** n (0 allowed)
REAL	1	**Trigger time** [s] of trigger event 1
REAL	1	**Pre trigger time** [s] of trigger event 1
REAL	1	**Post trigger time** [s] of trigger event 1
…		
REAL	1	**Trigger time** [s] of trigger event n
REAL	1	**Pre trigger time** [s] of trigger event n
REAL	1	**Post trigger time** [s] of trigger event n

3.8 The Sample Reduction Block SRBLOCK

The SRBLOCK serves to describe a sample reduction for a channel group, i.e. an alternative sampling of the signals in the channel group with a usually lower number of sampling points. There can be several sample reductions for the same channel group which are stored in a forward linked list of SRBLOCKs starting at the CGBLOCK.

Each SRBLOCK describes a sample reduction calculated with a specific time interval. Roughly explained, the reduction divides the time axis into intervals of the given fixed length and compresses all samples within one interval to three values: mean, minimum and maximum value. Each sample reduction applies to all signals of its parent channel group. For details of calculating and storing a sample reduction for a channel group, see below.

Sample reductions are only "convenience" information to accelerate graphical display or offline analysis of the signal data. It may be omitted without any loss of information. If not present, it can be added later by offline processing of the file.

3.8.1 Block structure of SRBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "SR"
UINT16	1	**Block size** of this block in bytes (entire SRBLOCK)
LINK	1	Pointer to **next sample reduction block** (SRBLOCK) (NIL allowed)
LINK	1	Pointer to the **data block** for this sample reduction (see below)
UINT32	1	**Number of reduced samples** in the data block.
REAL	1	**Length of time interval [s]** used to calculate the reduced samples.

3.8.2 Motivation for Sample Reduction

When drawing a graphical time diagram of a channel that has a large number of sample points, the application usually has two options:

1. Print each single sample point into the graph. Since there are more sample points than pixels for the time axis, several sample points will be drawn onto the same pixel position for the time axis. Assuming that the time axis is horizontal, we thus will not get a point in time, but a vertical line. The advantage here is that this line shows the minimum and maximum value that occurred for the time interval represented by the pixel. However, drawing each single sample point may be quite slow, especially for a huge number of samples.
2. Print only one sample point per time pixel. Thus we really get a point for each pixel and accelerate the drawing. On the other hand, the local minimum and maximum values that occurred in the time interval of the pixel will not be visible any longer.

The idea of the sample reduction is to provide a trade-off between the two methods. For each pixel of the time axis three values are required: the minimum, maximum and mean value for the time interval that will be reduced to a pixel position on screen. Obviously, the drawing application could calculate these values each time again from the row of sample points. With the SRBLOCK, these values can be calculated only once and stored in the MDF file.

The sample reduction applies to all channels of a channel group because the sample points of the channels in a record must have the same time stamp. The time interval represented by a pixel can vary due to different screen resolutions, or when the graph is zoomed. Thus, a number of sample reductions can be stored for different interval lengths. The drawing application then can determine the best fitting interval length when drawing the graph.

3.8.3 Layout of Reduced Samples

For a given channel group, a linked list of sample reduction blocks (SRBLOCKs) can be added. Each SRBLOCK uses a different time interval for the sample reduction and points to a special data block containing the reduced samples.

Note: the data block referenced by a SRBLOCK must not contain other records than the records of the reduced signal data! Thus a record ID is not necessary.

For the sample reduction, the time axis will be divided into time intervals of equal length. Each time interval that contains at least one sample point will be stored as reduced sample for the SRBLOCK. For each channel, the reduced sample contains its minimum, maximum and mean value of all sample points within the given time interval.

Since three values for each channel must be stored for a reduced sample, three records are stored subsequently (denoted as record 1, record 2 and record 3). The record layout of the three records is equal to the record layout defined by the parent channel group (but without record ID). However, for each of the three records, the meaning of the signal is different, see table below.

	Normal numeric channels	**Time channel**
Record 1	Mean value of all samples for this channel within the given time interval	Start time t of interval. The interval is defined as [t, t+L[, where L is the fixed length of the intervals for this SRBLOCK.
Record 2	Minimum value of all samples for this channel within the given time interval	Minimum raster value $\Delta > 0$ for the given interval, or 0 if no valid raster value available for the interval. The raster value of a sample is the calculated difference between the time stamp of the current sample and of the previous one. The raster value is undefined if the current sample is the very first sample (record index 0). Mathematically the minimum raster value of the interval [t, t+L[is defined as minimum of all $\Delta_i = t_i - t_{i-1}$ for all record indexes $i > 0$ so that t_i is in [t, t+L[. If there is no single valid Δ_i for this interval exists, the value must be set to 0.
Record 3	Maximum value of all samples for this channel within the given time interval	Maximum raster value $\Delta > 0$ for the given interval, or 0 if no valid raster value available for the interval (see explanation for Record 2).

Please note that the determination of minimum, maximum and mean value considers the raw values and not the physical values.

For non-numerical signal data types, a sample reduction does not make much sense because these values cannot be represented in a graph. In case both numerical and non-numerical values occur within the same channel group, the minimum, maximum and mean value for the channel with the non-numerical signal data type is not defined. As good practice simply store the non-numerical values of the first sample point within the interval for all three records of the reduced sample.

3.9 The Data Group Block DGBLOCK

The DGBLOCK gathers information and links related to its data block. Thus the branch in the tree of MDF blocks that is opened by the DGBLOCK contains all information necessary to understand and decode the data block referenced by the DGBLOCK.

The DGBLOCK can contain several channel groups. In this case the MDF file is "unsorted". If there is only one channel group in the DGBLOCK, the MDF file is "sorted" (please refer to chapter 4 for details).

3.9.1 Block structure of DGBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "DG"
UINT16	1	**Block size** of this block in bytes (entire DGBLOCK)
LINK	1	Pointer to **next data group block** (DGBLOCK) (NIL allowed)
LINK	1	Pointer to **first channel group block** (CGBLOCK) (NIL allowed)
LINK	1	Pointer to **trigger block** (TRBLOCK) (NIL allowed)
LINK	1	Pointer to the **data block** (see separate chapter on data storage)
UINT16	1	**Number of channel groups** (redundant information)
UINT16	1	**Number of record IDs** in the data block 0 = data records without record ID 1 = record ID (UINT8) before each data record 2 = record ID (UINT8) before and after each data record
UINT32	1	Reserved

3.10 The Channel Group Block CGBLOCK

The CGBLOCK contains a collection of channels which are stored in a record, i.e. which have the same time sampling.

3.10.1 Block structure of CGBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "CG"
UINT16	1	**Block size** of this block in bytes (entire CGBLOCK)
LINK	1	Pointer to **next channel group block** (CGBLOCK) (NIL allowed)
LINK	1	Pointer to **first channel block** (CNBLOCK) (NIL allowed)
LINK	1	Pointer to channel group **comment** text (TXBLOCK) (NIL allowed)
UINT16	1	**Record ID**, i.e. value of the identifier for a record if the DGBLOCK defines a number of record IDs > 0
UINT16	1	**Number of channels** (redundant information)
UINT16	1	**Size of data record** in Bytes (without record ID), i.e. size of plain data for a each recorded sample of this channel group
UINT32	1	**Number of records** of this type in the data block i.e. number of samples for this channel group
LINK	1	Pointer to **first sample reduction block** (SRBLOCK) (NIL allowed) *Valid since version 3.30. Default value: NIL.*

3.11 The Channel Block CNBLOCK

The channel block contains detailed information about a signal and how it is stored in the record.

3.11.1 Block structure of CNBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "CN"
UINT16	1	**Block size** of this block in bytes (entire CNBLOCK)
LINK	1	Pointer to **next channel block** (CNBLOCK) of this channel group (NIL allowed)
LINK	1	Pointer to the **conversion formula** (CCBLOCK) of this signal (NIL allowed)
LINK	1	Pointer to the **source-depending extensions** (CEBLOCK) of this signal (NIL allowed)
LINK	1	Pointer to the **dependency block** (CDBLOCK) of this signal (NIL allowed)
LINK	1	Pointer to the channel **comment** (TXBLOCK) of this signal (NIL allowed)
UINT16	1	**Channel type** 0 = data channel 1 = time channel for all signals of this group (in each channel group, exactly one channel must be defined as time channel) The time stamps recording in a time channel are always relative to the start time of the measurement defined in HDBLOCK.
CHAR	32	**Short signal name**, i.e. the first 31 characters of the ASAM-MCD name of the signal (end of text should be indicated by 0). If the name is longer than 31 characters, the complete name is contained in the TXBLOCK referenced by field "long signal name" (see below). In this case, the short signal name should be ignored in favour of the long name. ***Note:*** *The name of a signal in combination with the information about its source (CEBLOCK) should be unique within the MDF file because this name might be used as identifier for evaluation programs.*
CHAR	128	**Signal description** (end of text should be indicated by 0)
UINT16	1	**Start offset in bits** to determine the first bit of the signal in the data record. The start offset N is divided into two parts: a "Byte offset" (= N div 8) and a "Bit offset" (= N mod 8). The channel block can define an "additional Byte offset" (see below) which must be added to the Byte offset. The (total) Byte offset is applied to the plain record data, i.e. without record ID. It determines the first Byte that contains bits of the signal value. The bit offset is used to determine the LSB within the Bytes for the signal value. For more details and an example, please refer to section *4.3 Reading Signal Values from a Data Record*.
UINT16	1	**Number of bits** used to encode the value of this signal in a data record
UINT16	1	**Signal data type** ***Note:*** *for 0-3 the default Byte order defined in IDBLOCK is used, for 9-16 the default Byte order is overruled!*
		0 = unsigned integer 1 = signed integer (two's complement) 2 = IEEE 754 floating-point format FLOAT (4 bytes) 3 = IEEE 754 floating-point format DOUBLE (8 bytes) — *Default Byte order from IDBLOCK*
		4 = VAX floating-point format (F_Float) 5 = VAX floating-point format (G_Float) 6 = VAX floating-point format (D_Float) — *obsolete*
		7 = String (NULL terminated) 8 = Byte Array (max. 8191 Bytes, constant record length!)
		9 = unsigned integer 10 = signed integer (two's complement) 11 = IEEE 754 floating-point format FLOAT (4 bytes) 12 = IEEE 754 floating-point format DOUBLE (8 bytes) — *Big Endian (Motorola) Byte order*

Data type	Count	Description	
		13 = unsigned integer 14 = signed integer (two's complement) 15 = IEEE 754 floating-point format FLOAT (4 bytes) 16 = IEEE 754 floating-point format DOUBLE (8 bytes)	*Little Endian (Intel) Byte order*
BOOL	1	**Value range valid** flag: TRUE = both the minimum and maximum raw value that occurred for this signal in the given time range are known and stored in the next two fields FALSE = the next two fields for minimum and maximum signal value are not valid	
REAL	1	**Minimum signal value** that occurred for this signal (raw value) (only valid if raw value range is known)	
REAL	1	**Maximum signal value** that occurred for this signal (raw value) (only valid if raw value range is known)	
REAL	1	**Sampling rate** for a virtual time channel. Unit [s] Can be used to define a so-called "virtual time channel", i.e. a time channel where the number of bits is equal to zero. In this case, the time stamps are not stored in the data record but determined by the record index and the sampling rate. ***Note:*** *Not to be applied for value channels or real time channels.*	
LINK	1	Pointer to TXBLOCK that contains the ASAM-MCD **long signal name** (NIL allowed). See remarks for short signal name above. *Valid since version 2.12. Default value: NIL.*	
LINK	1	Pointer to TXBLOCK that contains the signal's **display name** (NIL allowed) *Valid since version 3.00. Default value: NIL.*	
UINT16	1	**Additional Byte offset** of the signal in the data record (default value: 0). The additional Byte offset is applied to the plain record data, i.e. without record ID. It determines the Byte from with to apply the start offset. ***Note:*** *to ensure compatibility, this field should only be used if the start offset in bits is not sufficient to express the start position, i.e. if the index of the first Byte is ≥ 8192* . *Valid since version 3.00. Default value: 0.*	

Signals of 1 to 64 bits are supported both for unsigned and signed integers. For floating-point numbers compliant with the IEEE 754 standard, signals with 32 or 64 bits (corresponding to single and double) are supported. This applies both to data and time channels.

Note: To determine the signal data type, the field "number of bits" must be evaluated, too.

For the data type String and Byte Array the number of bits must be a multiple of 8 and can be in the range from 8 up to 65528 bits.
The start offset of Floating point signals or String and Byte Arrays must be a multiple of 8 (1-Byte alignment).
Integer data types must fit into 8 contiguous Bytes.

3.12 The Channel Conversion Block CCBLOCK

The data records only store raw values, i.e. the values as they are provided by the data source (e.g. ECU). The CCBLOCK serves to specify a conversion formula that can be used to convert these values into physical values with physical units.

3.12.1 Block structure of CCBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "CC"
UINT16	1	**Block size** of this block in bytes (entire CCBLOCK)
BOOL	1	**Physical value range valid** flag: TRUE = both the minimum and maximum (physical) value that occurred for this signal in the given time range are known and stored in the next two fields FALSE = the next two fields for minimum and maximum signal value are not valid
REAL	1	**Minimum physical signal value** that occurred for this signal (only valid if physical value range is known)
REAL	1	**Maximum physical signal value** that occurred for this signal (only valid if physical value range is known)
CHAR	20	**Physical unit** (string should be terminated with 0)
UINT16	1	**Conversion type** (formula identifier) 0 = parametric, linear 1 = tabular with interpolation 2 = tabular 6 = polynomial function 7 = exponential function 8 = logarithmic function 9 = rational conversion formula 10 = ASAM-MCD2 Text formula 11 = ASAM-MCD2 Text Table, (COMPU_VTAB) 12 = ASAM-MCD2 Text Range Table (COMPU_VTAB_RANGE) 132 = date (Based on 7 Byte Date data structure) 133 = time (Based on 6 Byte Time data structure) 65535 = 1:1 conversion formula (Int = Phys)
UINT16	1	**Size information** about additional conversion data (optional) Conversion types 0, 6, 7, 8, 9: number of parameters Conversion types 1, 2, 11, 12: number of value pairs Conversion type 10: number of characters
...	variable	**Additional conversion data** (optional) Parameters (for type 0, 6, 7, 8, 9) or table (for type 1, 2, 11, or 12) or text (for type 10), depending on the conversion formula identifier. See formula-specific block supplement.

The following specifies the conversion formulas or their representation in CCBLOCK in detail. Int refers to the raw (internal) value stored in the MDF file. Phys is the converted physical value.

3.12.2 CCBLOCK – Linear Function with 2 Parameters

$$Phys = Int * P2 + P1$$

Data type	Count	Description
REAL	1	P1
REAL	1	P2

3.12.3 CCBLOCK – Polynomial Function with 6 Parameters

$$Phys = \frac{P2 - [P4 * (Int - P5 - P6)]}{P3 * [Int - P5 - P6] - P1}$$

P6: only needed for twos complement representation:

For 1 Byte values P6* should be set to 256
For 2 Byte values P6* should be set to 65536
It is:
If Int > P6* / 2 – 1 Then P6 = P6*
If Int ≤ P6* / 2 – 1 Then P6 = 0

Data type	Count	Description
REAL	1	P1
REAL	1	P2
REAL	1	P3
REAL	1	P4
REAL	1	P5
REAL	1	P6

3.12.4 CCBLOCK – Tabular, with Interpolation

Base is a table with n break points. The int values are given in an increasing order (strictly monotonous increasing). If a value x is located between two table values, (tablevalue[i] ≤ x < tablevalue[i+1]), linear interpolation between physvalue[i] and physvalue[i+1] is used. Is x < than Int value 1, Physvalue 1 will be returned. If x is larger than or equal to the greatest Int value, Phys value n will be returned.

Data type	Count	Description
REAL	1	Int value 1
REAL	1	Phys value 1
...		
REAL	1	Int value n
REAL	1	Phys value n

3.12.5 CCBLOCK – Tabular

Basis is a table with n break points. The int values are given in an increasing order (strictly monotonous increasing). If a value x is located between two table values, the next lower phys value is used. If x is larger than or equal to the greatest Int value, Phys value n will be returned.

Data type	Count	Description
REAL	1	Int value 1
REAL	1	Phys value 1
...		
REAL	1	Int value n
REAL	1	Phys value n

3.12.6 CCBLOCK – Exponential

If P4 = 0:

$$Phys = \frac{\ln\left(\frac{(Int - P7) * P6 - P3}{P1}\right)}{P2}$$

If P1 = 0:

$$Phys = \frac{\ln\left(\frac{P3}{\frac{Int - P7}{P4}} - P6\right)}{P5}$$

Data type	Count	Description
REAL	1	P1
REAL	1	P2
REAL	1	P3
REAL	1	P4
REAL	1	P5
REAL	1	P6
REAL	1	P7

3.12.7 CCBLOCK – Logarithmic

If P4 = 0:

$$Phys = \frac{e^{\left(\frac{(Int - P7) * P6 - P3}{P1}\right)}}{P2}$$

If P1 = 0:

$$Phys = \frac{e^{\left(\frac{P3}{\frac{Int - P7}{P4}} - P6\right)}}{P5}$$

Data type	Count	Description
REAL	1	P1
REAL	1	P2
REAL	1	P3
REAL	1	P4
REAL	1	P5
REAL	1	P6
REAL	1	P7

3.12.8 CCBLOCK –Rational Conversion Formula

$$Phys = \frac{P1 * Int^2 + P2 * Int + P3}{P4 * Int^2 + P5 * Int + P6}$$

Attention: In contrast to ASAM-MCD 2MC Keyword RAT_FUNC, in MDF the conversion is vice versa, i.e. from raw (internal) value to physical value!

Data type	Count	Description
REAL	1	P1
REAL	1	P2
REAL	1	P3
REAL	1	P4
REAL	1	P5
REAL	1	P6

3.12.9 CCBLOCK – ASAM-MCD2 Text formula

With this conversion a physical value is assigned to a given Int value by using a calculation rule which consists of a text formula. In the text formula the Int value is specified as X1.

Example: The text formula "X1 * X1 + 1" multiplies the Int value by itself and adds 1 to the result.
See also ASAM-MCD 2MC keyword FORMULA.

Data type	Count	Description
CHAR	256	Text formula (string should be terminated by 0)

3.12.10 CCBLOCK – ASAM-MCD2 Text Table

With this conversion a given Int value a text is assigned within a table. This can be used to assign bit values or array of bits to verbal descriptions (for example "Gear 1", "Gear 2, "Gear 3", "On", "Off").

See also ASAM-MCD 2MC keyword COMPU_VTAB.

Data type	Count	Description
REAL	1	Int Value 1
CHAR	32	Assigned Text 1 (string should be terminated by 0)
...		
REAL	1	Int Value n
CHAR	32	Assigned Text n (string should be terminated by 0)

3.12.11 CCBLOCK – ASAM-MCD2 Text Range Table

With this conversion, a text is assigned to a given value range, i.e. the given text is displayed for all values `lower_range <= value < upper_range` for float and `lower_range <= value <= upper_range` for non-float values. The first entry contains the default range (i.e. the DEFAULT string that is to be used in case that there are no other assignments).
This can be used to assign float ranges to verbal descriptions (for example "low", "regular", "high").
See also ASAM-MCD 2MC keyword COMPU_VTAB_RANGE.

Note: Overlapping ranges may not be declared

Data type	Count	Description
REAL	1	Undefined (to be ignored)
REAL	1	Undefined (to be ignored)
LINK	1	Pointer to TXBLOCK that contains the DEFAULT text
REAL	1	Lower range 1
REAL	1	Upper range 1
LINK	1	Pointer to TXBLOCK that contains text 1
...		
REAL	1	Lower range n
REAL	1	Upper range n
LINK	1	Pointer to TXBLOCK that contains text n

3.12.12 CCBLOCK – Data Structure Date

This conversion formula is only valid for the data type Byte Array consisting of 7 Byte. Additional parameters from the CCBLOCK are not necessary. This representation corresponds to the data type "Date" of the CiA-CANopen specification "Application Layer and Communication Profile", Version 4.0. Within the Byte Array the Date is specified in the following way:

Data type	Identifier	Description
UINT16	ms	Bit 0 .. Bit 15: Milliseconds (0 .. 59999)
BYTE	min	Bit 0 .. Bit 5: Minutes (0 .. 59) Bit 6 .. Bit 7: Reserved
BYTE	hour	Bit 0 .. Bit 4: Hours (0 .. 23) Bit 5 .. Bit 6: Reserved Bit 7: 0 = Standard time, 1 = Summer time
BYTE	day	Bit 0 .. Bit 4: Day (0 .. 31) Bit 5 .. Bit 7: Day of week (1 = Monday … 7 = Sunday)
BYTE	month	Bit 0 .. Bit 5: Month (1 = January .. 12 = December) Bit 6 .. Bit 7: Reserved
BYTE	year	Bit 0 .. Bit 6: Year (0 .. 99) Bit 7: Reserved

3.12.13 CCBLOCK – Data Structure Time

This conversion formula is only valid for the data type Byte Array consisting of 6 Byte. Additional parameters from the CCBLOCK are not necessary. This representation corresponds to the data type "Time" of the CiA-CANopen specification "Application Layer and Communication Profile", Version 4.0. Within the Byte Array the Time is specified in the following way:

Data type	Identifier	Description
UINT32	ms	Bit 0 .. Bit 27: Number of Milliseconds since midnight of Jan 1st 1984 Bit 28 .. Bit 31: Reserved
BYTE	days	Bit 0 .. Bit 15: Number of days since Jan 1st 1984 (can be 0)

3.13 The Channel Dependency Block CDBLOCK

A signal may be a composition of other signals, e.g. for a map/curve or a structure. When saving such a compound signal, this can be modelled in MDF by a CNBLOCK for the compound signal itself having a link to a CDBLOCK. The CDBLOCK then lists all signal dependencies, i.e. all signals the compound signal consists of.

Note: A signal dependency can be a compound signal again.

3.13.1 Block structure of CDBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "CD"
UINT16	1	**Block size** of this block in bytes (entire CDBLOCK)
UINT16	1	**Dependency type** 0 = no dependency 1 = linear dependency (vector) 2 = matrix dependency (signals for each matrix line in a separate CGBLOCK) *Since version 3.01:* 256 + N = N-dimensional dependency (size of each dimension is listed after the variable length LINK list, dependent signals can be stored in same CGBLOCK)
UINT16	1	Total **number of signals dependencies** (m)
LINK	1	Pointer to the **data group** block (DGBLOCK) **of signal dependency 1**
LINK	1	Pointer to the **channel group** block (CGBLOCK) **of signal dependency 1**
LINK	1	Pointer to the **channel** block (CNBLOCK) **of signal dependency 1**
…		
LINK	1	Pointer to the **data group** block (DGBLOCK) **of signal dependency m**
LINK	1	Pointer to the **channel group** block (CGBLOCK) **of signal dependency m**
LINK	1	Pointer to the **channel** block (CNBLOCK) **of signal dependency m**
UINT16	1	Optional: **size of dimension 1** for N-dimensional dependency
	…	*Optional fields for N-dimensional dependency valid since version 3.01*
UINT16	1	Optional: **size of dimension N** for N-dimensional dependency

The CDBLOCK basically is a list of references to the "elements" of the compound signal. Each element is an individual signal which is specified by a LINK triple to its DGBLOCK, CGBLOCK and CNBLOCK.

For N ≥ 2 dimensions, the elements are listed in a "flat" list which must be interpreted line-oriented.

Example

A 3 x 2 matrix $M=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$ would be stored as follows: $a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}$

More generally, a i x j x k Matrix would be listed as $a_{111},...,a_{11i},a_{121},...,a_{12i},...,a_{1ji},a_{211},...,a_{kji}$

For N = 2, either each line of the matrix will be stored in a separated CGBLOCK (e.g. for Dependency type = 2 a_{11}, a_{12}, a_{13} will be stored in one CGBLOCK and a_{21}, a_{22}, a_{23} in another one), or the size of each dimension will be listed after the list of references (e.g. for Dependency type = 257, the last two elements have values 3 and 2).
For N > 2, only the latter possibility exists.

Note:
The values in the data record that are referenced by the CNBLOCK of the compound signal can be arbitrary. For simplicity we recommend to reference the signal values of the first dependent signal (first element of map/curve or structure). As a consequence, then the CNBLOCK of the compound signal and the CNBLOCK of the first dependency must be in the same channel group (CGBLOCK), and both CNBLOCKs have the same start offset and number of bits.

3.14 The Channel Extension Block CEBLOCK

The CEBLOCK serves to specify source specific properties of a channel, e.g. name of the ECU or name and sender of the CAN message

3.14.1 Block structure of CEBLOCK

Data type	Count	Description
CHAR	2	**Block type identifier**, always "CE"
UINT16	1	**Block size** of this block in bytes (entire CEBLOCK)
UINT16	1	**Extension type** identifier 2 = DIM (DIM block supplement) 19 = Vector CAN (Vector CAN block supplement)
...	variable	**Additional fields** depending on the extension type. See extension-specific block supplements.

Note:
- for CAN signals, type Vector CAN (19) should be used
- for ECU signals (CCP, XCP), type DIM (2) should be used

3.14.2 DIM block supplement

Data type	Count	Description
UINT16	1	Number of module
UINT32	1	Address
CHAR	80	Description
CHAR	32	Identification of ECU

3.14.3 Vector CAN block supplement

Data type	Count	Description
UINT32	1	Identifier of CAN message
UINT32	1	Index of CAN channel
CHAR	36	Name of message (string should be terminated by 0)
CHAR	36	Name of sender (string should be terminated by 0)

4 MDF Data Format

The actual values of the signals, i.e. the time samples for the measurement row, are encoded in the data block. A data block is always related to a data group and may only contain signal values for channels respectively channel groups out of this data group.

Signal values that are acquired at the same instants of time are collected and encoded in so-called data records. A data block thus consists of an arbitrary number of data records. Each data record normally also contains the time stamp it was recorded.

The descriptions of the channels whose signal values are collected in the same data record are stored in a channel group. Hence, a channel group lists only channels whose signal values are acquired simultaneously and stored in the same data record.

In general, a MDF file can either be "sorted" or "unsorted". A sorted MDF file allows faster read access for individual signal values, whereas for recording measurement data it usually is easier and faster to write an unsorted MDF file. Thus, as a strategy, a recording tool may write an unsorted MDF file, possibly sorting it after the recording has finished. On the other hand, an analysis tool might sort the MDF file once before opening it.

A sorted MDF file requires that each data block contains only data records of the same channel group, i.e. the data block can contain only one channel group.

In case of an unsorted MDF file, data records of different channel groups may be included. To be able to distinguish the data records for an unsorted MDF file, each is preceded and optionally followed by a record ID consisting of the data type UINT8. For a sorted MDF file, the record ID can be (and usually is) omitted.

The record structure is defined by the index of the first byte containing bits of the signal (byte part of the start offset and additional Byte offset), the bit offset in the LSB and the number of bits specified for each signal of the channel group in its CNBLOCK:

- For the data type String or Byte Array the number of bits must be a multiple of 8 (only whole Bytes). In addition to that a channel of the data type String or Byte Array has to start on a Byte limit (Start offset in bits must be a multiple of 8).
- For floating-point data types, 32 or 64 bits (4 or 8 Bytes, float and double) must be used, depending on the float type, and they must start at a Byte limit (Start offset in bits must be a multiple of 8).
  Note: The 10 Byte double (extended precision) channel data type has been declared obsolete because the IEEE 754 standard does not specify it, and because it is not supported by any known tool.
- For Integer data types, an arbitrary number of bits are allowed, i.e. you may encode a signal value with a range of [0-7] in 3 bits only. However, the Integer data type must fit into 8 contiguous Bytes. Note: some applications may only accept up to 32 bits for an Integer value or expect that it fits into 4 contiguous Bytes.

Up to MDF 3.10, signals with a number of bits ≤ 7 must not exceed a Byte limit, and signals with a number of bits ≥ 8 have to start at a Byte limit. Starting with MDF version 3.1, these restrictions have been removed for Integer signals, but still an Integer signal must fit into 8 Bytes.

4.1 Data Format for Sorted MDF Files

All the signal values of a channel group constitute a data record. For a sorted MDF file there is exactly one channel group for each data group available. Thus all records in the data block have the same record format. As a consequence, the record ID can be omitted.

The data records inside a data block are listed in the sequence of the time stamps.

Note: If the records are not listed based on the time stamps, this should be interpreted as an error.

4.2 Data Format for Unsorted MDF Files

In contrast to a sorted MDF file, in an unsorted MDF file data records from different channel groups may be stored sequentially in the data block. Each record is clearly related to a channel group by means of a record ID (of data type UINT8) at the beginning of the record and optionally after the end of each record (see CGBLOCK, Record ID). The record structure itself is identical to the one for a sorted MDF file.

The data records inside a data block are listed for each record type in the sequence of their time stamps.

Note: If the records are not listed based on the time stamps, this should be interpreted as an error.

Reading a data block of an unsorted MDF file, the records have to be read sequentially from the start of the data block. For each record, the record ID must be checked and the size of the record for the respective channel group must be considered.

4.3 Reading Signal Values from a Data Record

The structure of the data record is defined by the following elements of CNBLOCK:

- signal data type
- number of bits
- start offset
- additional Byte offset

In addition, we also have to consider the number of Bytes used for the record ID, specified in DGBLOCK, and – if not specified by the signal data type yet – the default Byte order / Floating point format specified in IDBLOCK.

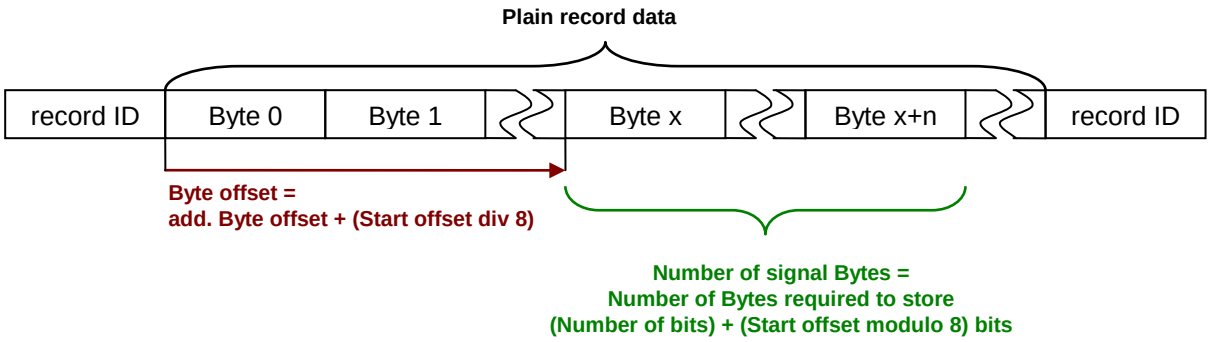
The start position of the signal is always counted from the start of the plain record data, i.e. without the Bytes used for the record ID. So, if a record ID is present, we have to skip its Bytes at the start of the record. Then we determine the first Byte of the "signal Bytes", i.e. the first Byte of the minimum range of Bytes that contains all bits of the signal value.

Next, we copy the signal Bytes to a memory buffer. Here we have to consider the Byte order specified by the signal data type, or, if not specified there, by the default Byte order of the file. If the specified Byte order is different to the memory system our evaluation program runs on, we have to adapt the Byte order. This means that we have to swap the required Bytes when copying them to a memory buffer (see example below).

Now we apply the bit offset to the memory buffer, i.e. we right-shift the bits by bit offset. As a result, the LSB of the signal becomes the LSB of the first Byte of the buffer. We now can interpret the number of bits for the signal as value, which possibly requires masking unused bits.

Recapitulating, the steps to de-code the record value are

1. Skip record ID
2. Apply Byte offset = additional Byte offset + (Start offset *div* 8)
3. Copy signal Bytes to memory buffer
4. Possibly swap signal Bytes (depending on Byte order)
5. Apply bit offset = right shift by (start offset *modulo* 8) bits
6. Mask unused bits
7. Read and interpret the value in memory buffer



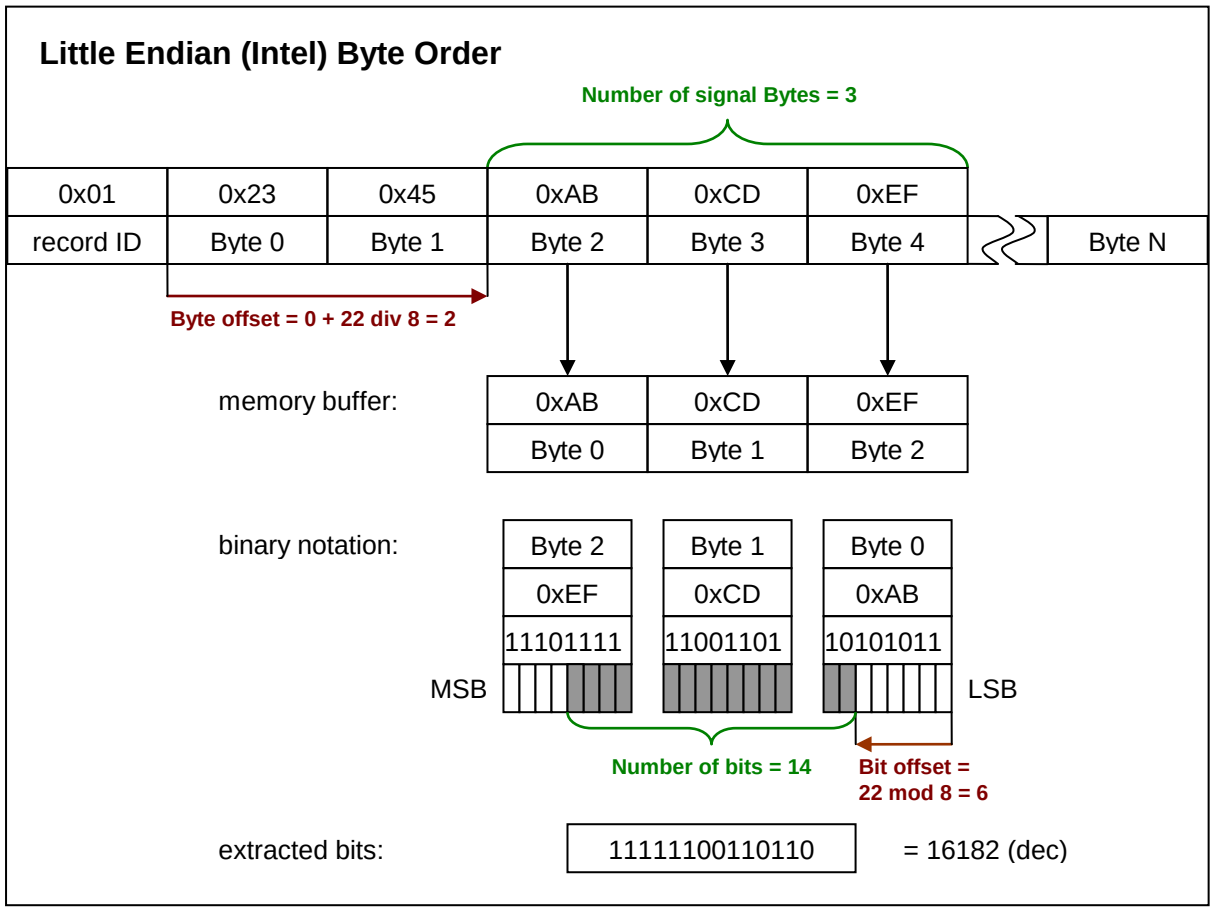
4.3.1 Example 1: Little Endian (Intel) Byte Order

We assume a (unsorted) MDF file with Little Endian (Intel) default Byte order, which we evaluate on an Intel system. The CNBLOCK of the signal we want to de-code has the following values:

- signal data type = 0 (=> unsigned integer with default Byte order)
- number of bits = 14
- start offset = 22
- additional Byte offset = 0

The number of Bytes used for the record ID is 1 (specified in DGBLOCK), so the very first Byte of each record contains its record ID.

As can be seen in the following graphic, after extracting the relevant bits for the signal, we get a value of 16182 in decimal notation.



4.3.2 Example 2: Big Endian (Motorola) Byte Order

For this example we consider exactly the same settings as for Example 1, expect that the signal data type now is 9 = unsigned integer with Big Endian (Motorola) Byte order.

The Byte order specified in the signal data type overrules the default Byte order of the file, because the file might have been written by a program running on an Intel system, but the signals were acquired from an ECU with a Motorola processor.

The process to de-code the signal value is exactly the same, with the only difference that when copying the signal Bytes to the memory buffer, we have to swap the Bytes (Byte 0 with Byte (N-1), Byte 1 with Byte (N-2), etc.). The application of the bit offset to the memory buffer then is identical (usually implemented by a left shift operation).