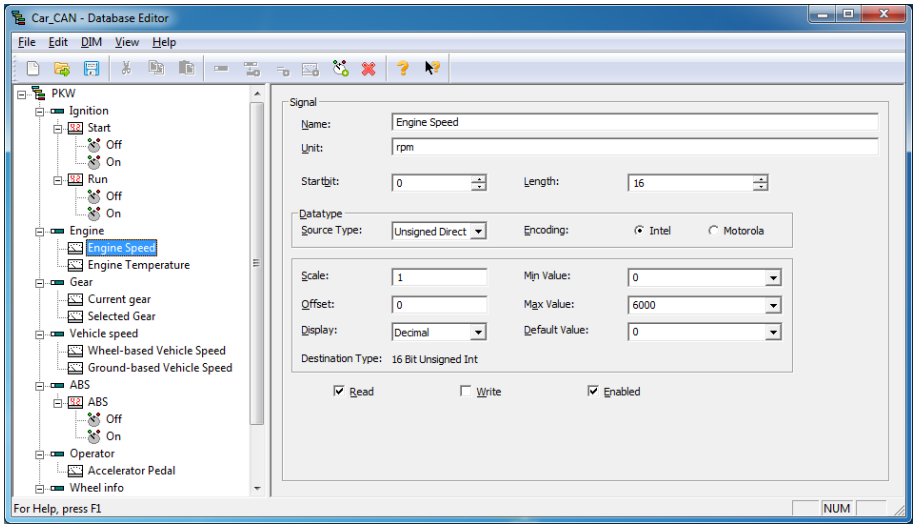
Ixxat®
BY HMS NETWORKS

DIMEditor

USER MANUAL

ENGLISH



HMS Technology Center Ravensburg GmbH
Helmut-Vetter-Straße 2
88213 Ravensburg
Germany

Tel.: +49 751 56146-0
Fax: +49 751 56146-29
Internet: www.hms-networks.de
E-Mail: info-ravensburg@hms-networks.de

Support

For problems or support with this product or other HMS products please request support at www.ixxat.com/support.

Further international support contacts can be found on our webpage www.ixxat.com

Copyright

Registered trademarks

Document number: 1.02.0133.30000-DIMEdit
Version: E-2.31

Contents

Chapter 1

Overview

1.1 The Data Interpreter Module (DIM)

The Data Interpreter Module enables the data transmitted in a CAN telegram to be **interpreted** as physical values or states.
The Data Interpreter Server forms the central element of the **Data Interpreter Module**. It receives the CAN telegrams and carries out the interpretation with a database determined by the user. This database contains **signal definitions** which describe the value and position within the CAN telegram, the format and the conversion factors of **signals**.
The Data Interpreter Module supports the following types of signals:

- **Analogue signals** in various codifications, as well as a conversion regulation based on a linear equation
- **Digital signals**, to which pairs consisting of value and state description are allocated.

During the interpretation phase, the bit patterns of the CAN telegram are converted into meaningful numerical values and thus provided as physical values or states.
The Data Interpreter Module consists of three program parts (Abb. 1.1):

- The **Data Interpreter Server** forms the central element. It receives CAN telegrams and carries out interpretation with a given database.
- With the **Database Editor**, the user can draw up and process Data Interpreter databases. The signal definitions are deposited in the DIM data base.
- The **Signal module** displays the signals and their current value

The following sections describe the basic principles which are a prerequisite for using the Data Interpreter Module. In section "Quick Start - Definition of an example database", an example database is drawn up with the aid of the Database Editor. Working through this section will provide a quick introduction to the Data Interpreter Module.

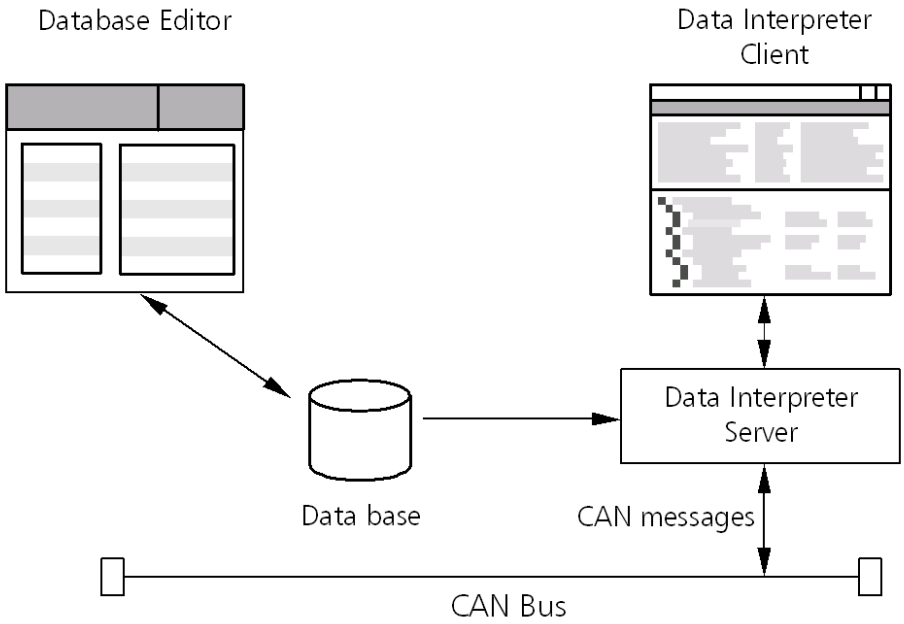


Figure 1.1: Data Interpreter Module components

Chapter 2

Data Interpreter Module basic principles

2.1 Functional principle

A consideration of the information transmitted by means of a serial bus system shows that a bit stream is involved. Only a specified protocol gives the bits a meaning and makes the information on the bus usable for the receiver.
Basically, two types of information are transmitted:

- **Protocol information** (describes the meaning of the remaining information (bits))
- **Data**

Protocol information is described by the Data Interpreter Module with so-called **Data Interpreter multiplexers** whereas contained data are described by **signal definitions**.
The Data Interpreter Module handles both information within one database.
The position of the bits carrying information is described by stating the start bit and bit length. Thus, the Data Interpreter Module cannot process any information which is distributed over unrelated fields in a bit stream.
In the following, the definition of signals and multiplexers is explained.

2.2 Signal definitions

A **signal definition** allocates a certain **meaning** to a field within a message (bit stream). A signal is specified by following attributes:

- Name
- Field (start bit and length)
- Codification
  - Byte-Order (INTEL/Motorola)
  - Data format
- Linear equation (Sizing factor and offset)

Possible data formats are:

- Unsigned binary (Unsigned integer, 1 - 64 bit)

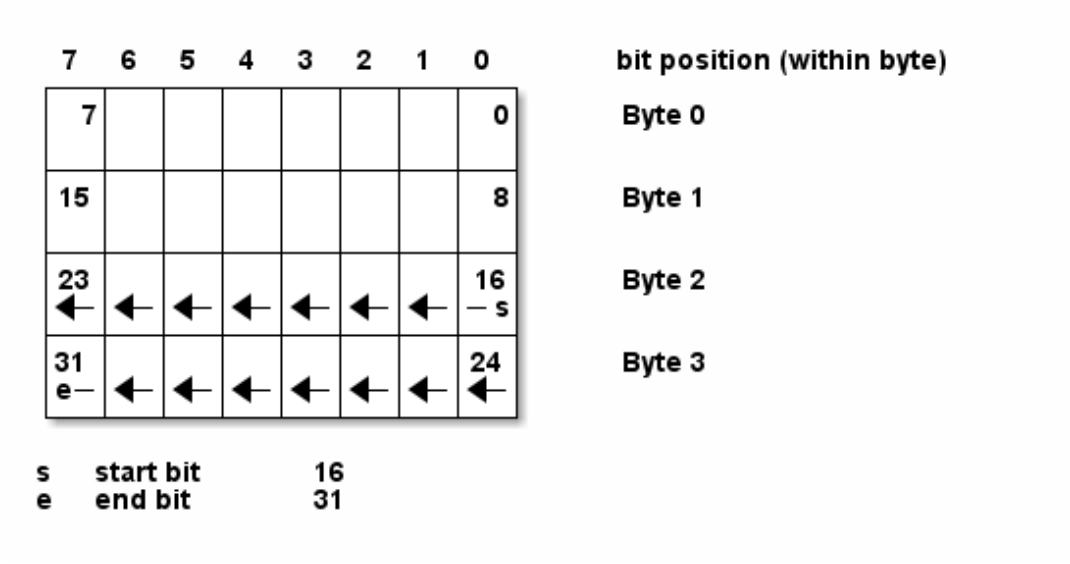


Figure 2.1: For Intel signals the start bit is at the LSB.

- Signed binary (2 - 64 bit) (Signed integer, 1's complement, 2's complement)
- Unsigned packed BCD digits (4 - 64 bit, 4 bit granularity)
- Signed packed BCD digits (8 - 64 bit, 4 bit granularity)
- IEEE floating point numbers (32 resp. 64 bit)

In order to enable pre-definition of signal values or checking of incoming signals, additional information is required which determines

- a minimum, maximum and default value and
- the access type of the signals (readable and/or writable).

The remaining settings are used to parameterise the output of the signals within the Signal module:

- colour
- output format (ASCII, binary, octal, decimal, hexadecimal, FormatStr).

2.2.1 Byte-Order (Endianess)

The byte order flag specifies which endianess is used (Intel or Motorola). This affects the direction in which the signal bits are extracted and the position of the start bit. The start bit position is specified in "Intel standard" notation for Intel coded signals and in "Motorola forward MSB" notation for Motorola encoded signals.
The start bit of Motorola signals is specified in "Motorola forward MSB" notation. Layout is the same as in "Motorola forward LSB" but start bit and end bit are exchanged. In "Motorola forward MSB" notation the start bit position is always lower than the end bit position.
Many tools use the "Motorola forward LSB" notation internally. Here the startbit position is higher than end bit position.

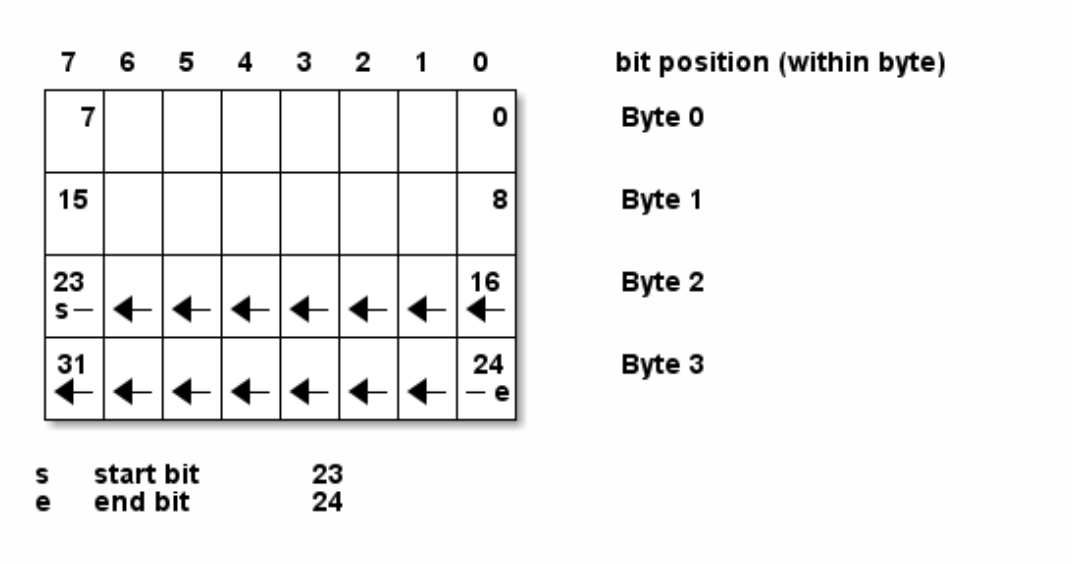


Figure 2.2: "Motorola forward MSB" notation

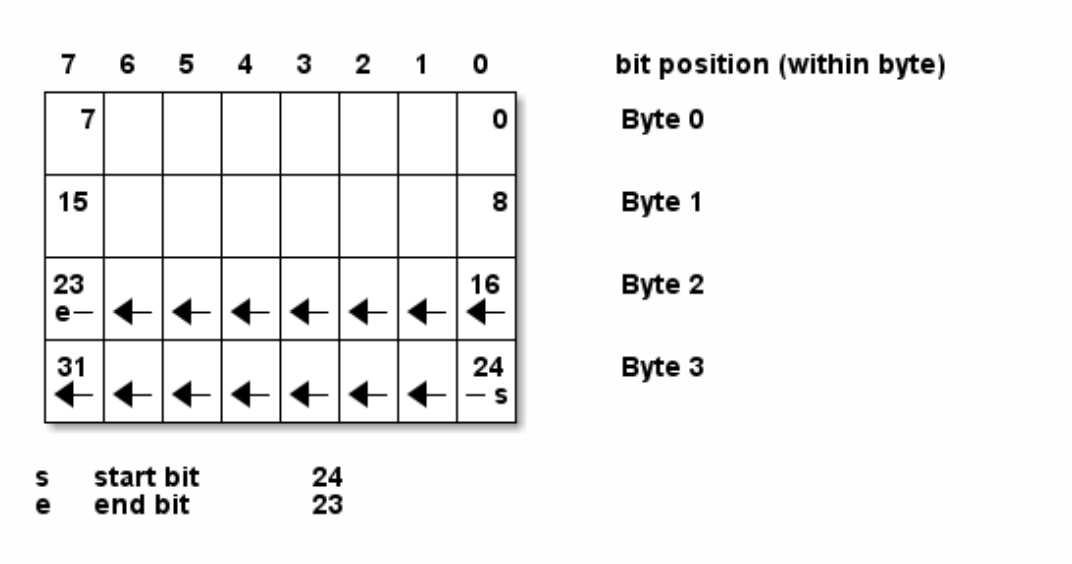


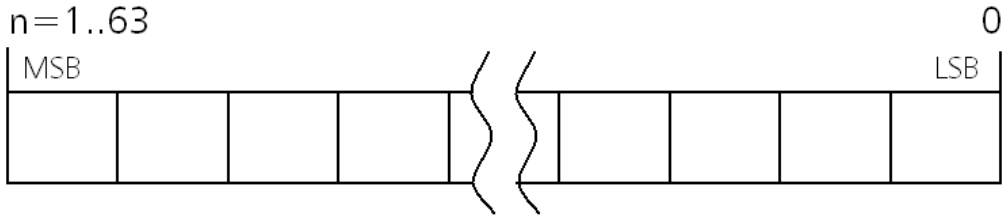
Figure 2.3: "Motorola forward LSB" notation

2.2.2 Digital signals

Digital signals are separated by analogue signals by applied additional state definitions.
Attention: The definition of states is only usable for signals with data type "Unsigned Binary".

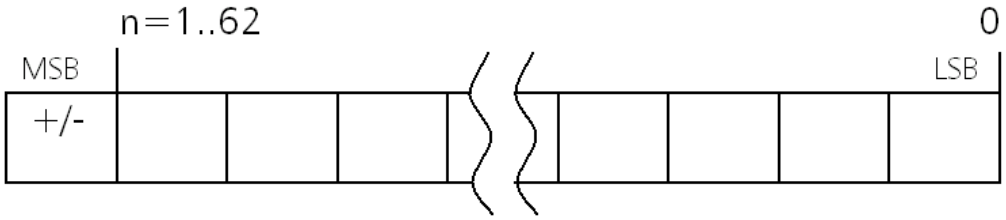
2.2.3 Data types

Unsigned binary



A signal of type "unsigned binary" is seen as absolute amount.

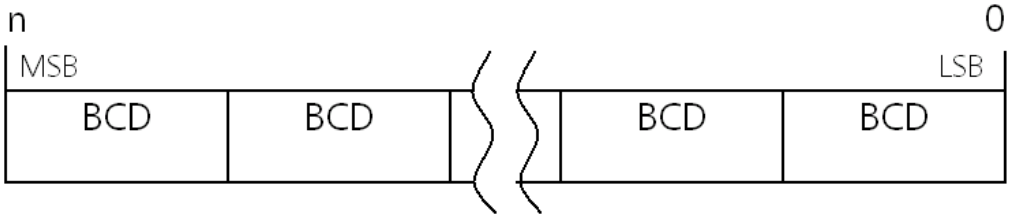
Signed binary



For Signed Binaries the sign is expected in the most significant bit (MSB).
There are three different supported codifications:

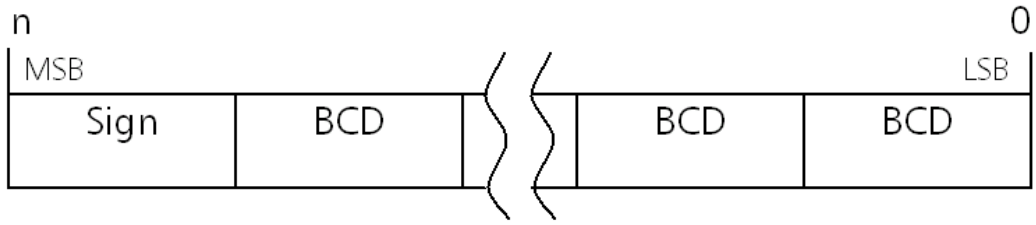
- Amount with sign
- 1’s complement
- 2’s complement

Unsigned BCD (4 - 64 bit, 4 bit granularity)



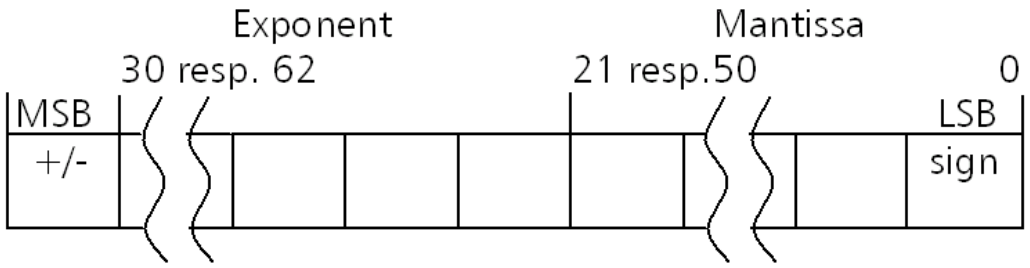
Four bits are combined to one BCD digit.

Signed BCD (8 - 64 bit, 4 bit granularity)



Four bits are combined to one BCD digit.
The sign consists of a 4 bit block, too. It is coded with 0000 (0 hex) for a positive sign and with 1000 (8 hex) for a negative sign.

IEEE floating point (32 or 64 bit)



IEEE floating points consists of a sign bit, a mantissa and an exponent. Two different IEEE formats are supported:

- IEEE 32 bit (22 bit mantissa, 7 bit exponent)
- IEEE 64 bit (51 bit mantissa, 10 bit exponent)

ASCII (8 - 64 bit, 8 bit granularity)

With the ASCII type data is interpreted as an ASCII string.

2.3 Comparison of integer data types

Value	Unsigned binary	Signed integer	1's compl.	2's compl.	Unsigned BCD	Signed BCD
+15	1111	-	-	-	-	-
+14	1110	-	-	-	-	-
+13	1101	-	-	-	-	-
+12	1100	-	-	-	-	-
+11	1011	-	-	-	-	-
+10	1010	-	-	-	-	-
+9	1001	-	-	-	1001	0000 1001
+8	1000	-	-	-	1000	0000 1000
+7	0111	0111	0111	0111	0111	0000 0111
+6	0110	0110	0110	0110	0110	0000 0110
+5	0101	0101	0101	0101	0101	0000 0101
+4	0100	0100	0100	0100	0100	0000 0100
+3	0011	0011	0011	0011	0011	0000 0011
+2	0010	0010	0010	0010	0010	0000 0010
+1	0001	0001	0001	0001	0001	0000 0001
+0	0000	0000	0000	0000	0000	0000 0000
-0	-	1000	1111	0000	-	1000 0000
-1	-	1001	1110	1111	-	1000 0001
-2	-	1010	1101	1110	-	1000 0010
-3	-	1011	1100	1101	-	1000 0011
-4	-	1100	1011	1100	-	1000 0100
-5	-	1101	1010	1011	-	1000 0101
-6	-	1110	1001	1010	-	1000 0110
-7	-	1111	1000	1001	-	1000 0111
-8	-	-	-	1000	-	1000 1000
-9	-	-	-	-	-	1000 1001
-10	-	-	-	-	-	-
-11	-	-	-	-	-	-
-12	-	-	-	-	-	-
-13	-	-	-	-	-	-
-14	-	-	-	-	-	-
-15	-	-	-	-	-	-

2.4 Data Interpreter multiplexers

A **Data Interpreter multiplexer** is an area within a bit stream (protocol information), which determines the meaning of the information remaining in the bit stream. In order to avoid redundant information within the database, a multiplexer is divided into two parts:

- The **multiplexer mask** determines the area by start bit and length
- The **multiplexer value** which determines a position within the area or alternatively an **else value**.

Example:

Assuming a protocol transmits a function number in the first byte, the value of this function number determines the meaning of the remaining bytes in the telegram.

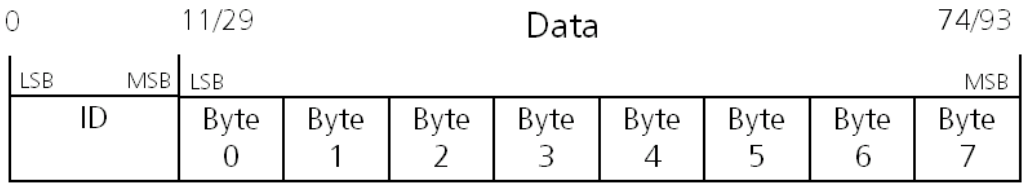


Figure 2.4: Structure of a CAN layer 2 message within the Data Interpreter Module

In order to interpret this protocol with the Data Interpreter Module, define a multiplexer mask which determines the first byte as area. With the aid of the multiplexer values below this, each function number can now be described. For example, you can assign a certain function name to the value 0.
Below the multiplexer value, the signal definitions for this function number are inserted and thus the meaning of the remaining data is defined.

2.4.1 Else value

Beneath a multiplexer mask, an alternative branch (**Else value**) is provided for in the Data Interpreter Module database.g
The interpretation rules of an Else value are applied when no appropriate multiplexer value is contained in the database.
No further multiplexers are permissible beneath an Else value.

2.4.2 Multiplexer independent signal definitions

Multiplexer independent signal definitions occupy a special position within a Data Interpreter database. They are arranged directly below the project and are evaluated first for the interpretation of a message.
One application for multiplexer independent signals is, for example, a protocol, which always deposits information, such as the address of a target node in the same area of a message.

2.5 The Data Interpreter Module and the CAN protocol

The Data Interpreter Module treats CAN telegrams like a bit stream and does not differentiate between ID and data area in the interpretation (Fig. 2.4) .
The CAN telegram consists of an 11 (29) bit wide identifier and a 64 bit (8 byte) wide data field. Theoretically, this total of 75(93) bits can be used to transmit any bit streams, but in practice a function is determined via the identifier and in the data field the parameters or data for this function are transmitted.

2.6 Interpretation of the CAN telegram

The interpretation basically goes through the following steps (Fig. 2.5):

- A CAN telegram is received and passed on for interpretation
- Message independent (global) signals are extracted from the telegram and interpreted.

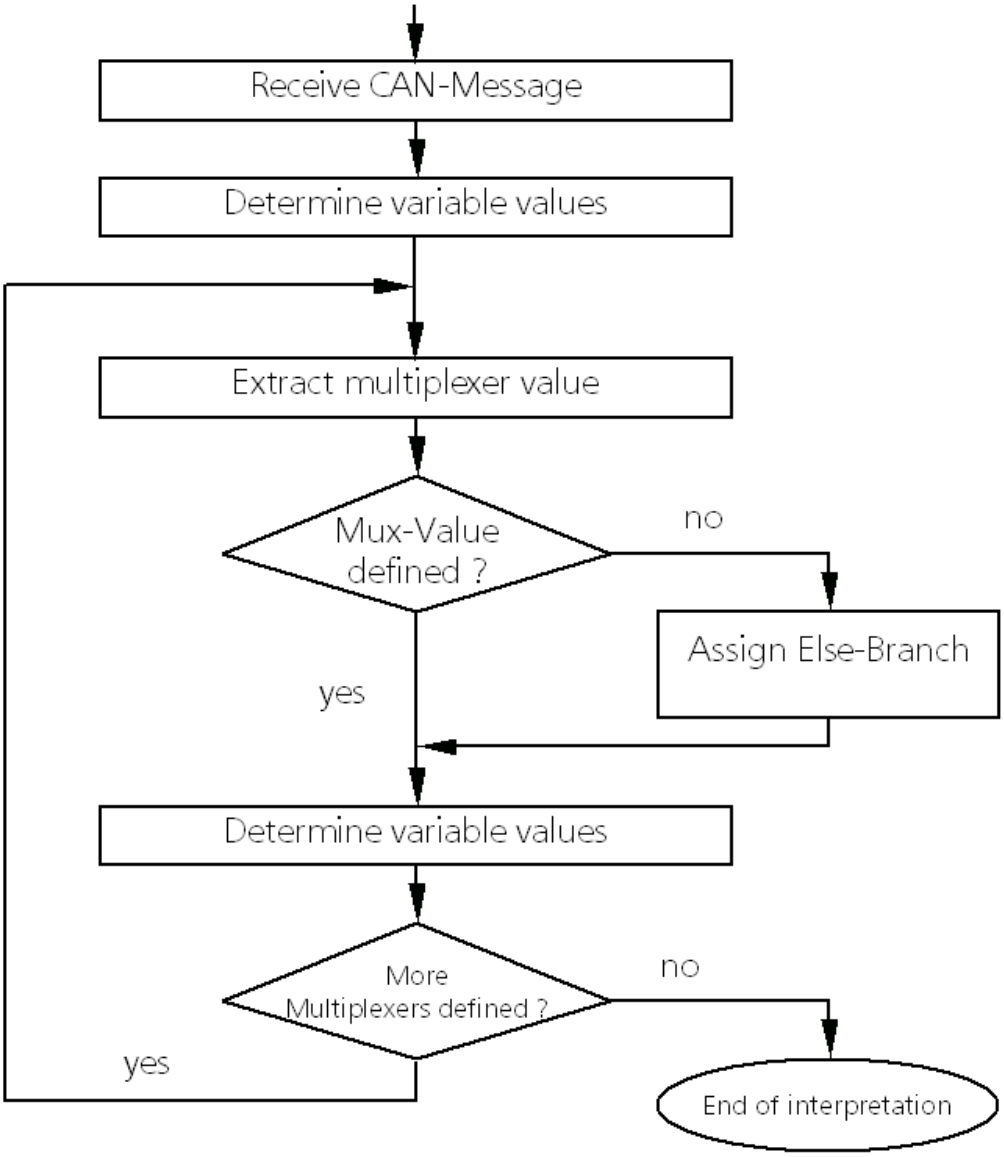


Figure 2.5: Interpretation of CAN-Messages in the Data Interpreter Module

- All message descriptions are matched in order. If a message description matches by id or mask/value the following steps are taken:
  - The first Mux mask is placed over the telegram and the Mux value is determined
  - If the corresponding Mux value is not defined, the assigned Else branch is used for the interpretation
  - The signals belonging to the current Mux value or Else branch are extracted from the telegram and interpreted
  - Then the next sub multiplexer is determined and the interpretations are carried out recursively.
- The same steps are taken for every message if a global Mux mask is defined

The interpretation process can be shown graphically by means of a tree structure. Fig. 2.6 shows the correlation in an exemplary way .

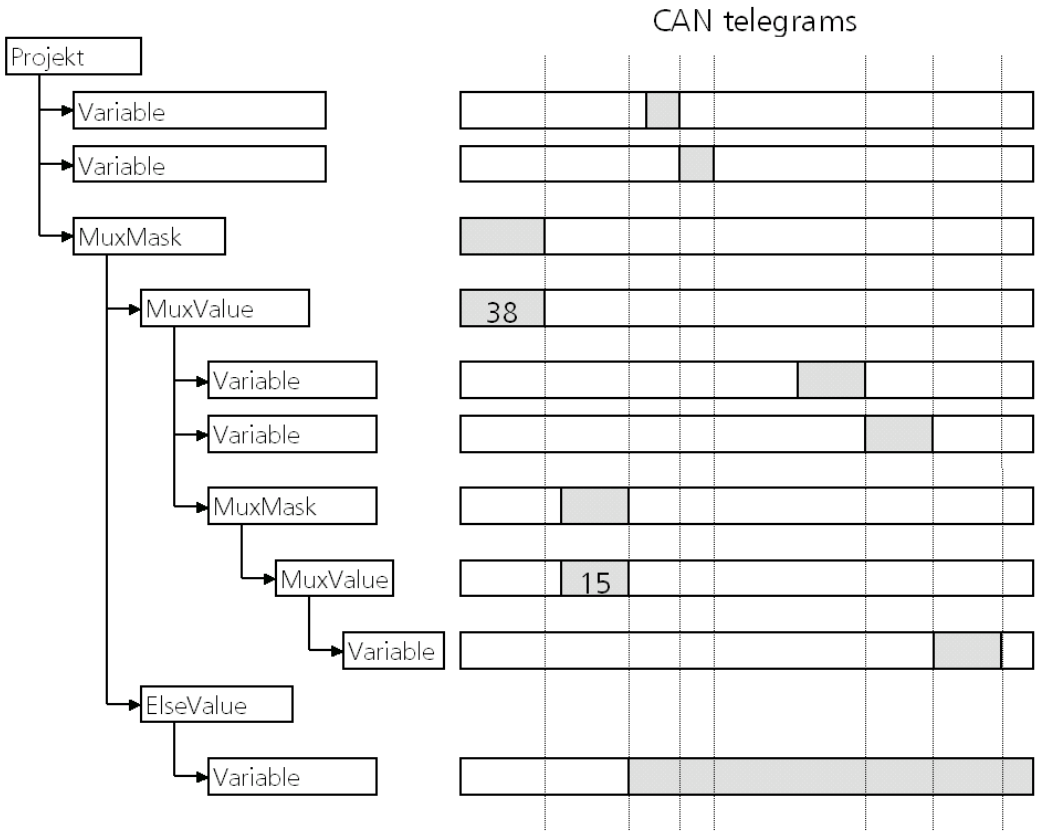


Figure 2.6: Structure of a Data Interpreter database

The project is situated on the root level with the project data belonging to it.
Under the root node the message objects are placed.
The multiplexer independent signals and the first multiplexer mask are situated one level deeper.
The multiplexer values (Mux values) below that with the multiplexer mask (Mux mask) together make up the multiplexer.
The given alternative branch (Else value) is brought in for the interpretation if none of the Mux values applies at the same level.
Below a Mux value, further multiplexers (Mask value pairs) can be situated.

2.7 Conversion rule for analogue signals

Analogue signals are translated from a raw value to a physical value by using a linear equation. Fig. 2.7 shows the translation process and the parameters that are involved.

Example

Within a CAN-Telegram the value of a temperature sensor is transmitted.
Within a CAN-Telegram the value of a temperature sensor is transmitted. The temperature sensor spits out values in the range of 0 to 4095 (12 Bit AD converter, 2^12 - 1 = 4095). These values corresponds to temperatures from -30 °C to +120 °C (Fig. 2.8).
The values for scale and offset describe the gradient and offset of the straight line. They are evaluated to:

```
Scale = (PhysMax - PhysMin) / (LogMax - LogMin)
```

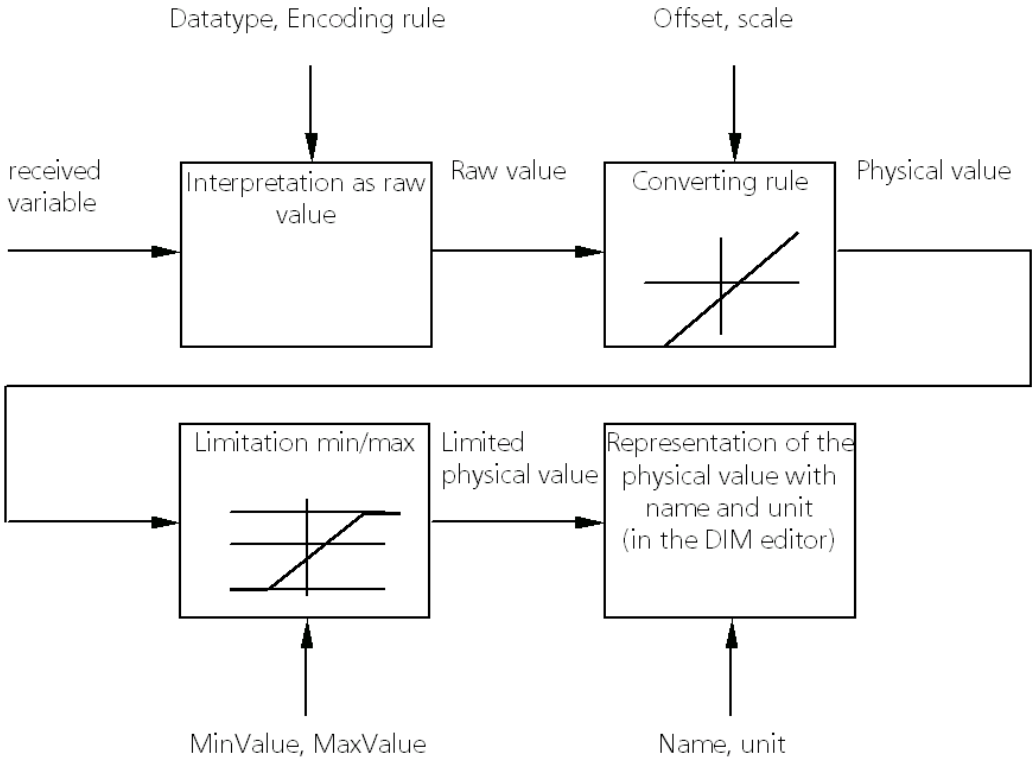


Figure 2.7: Translation of signals

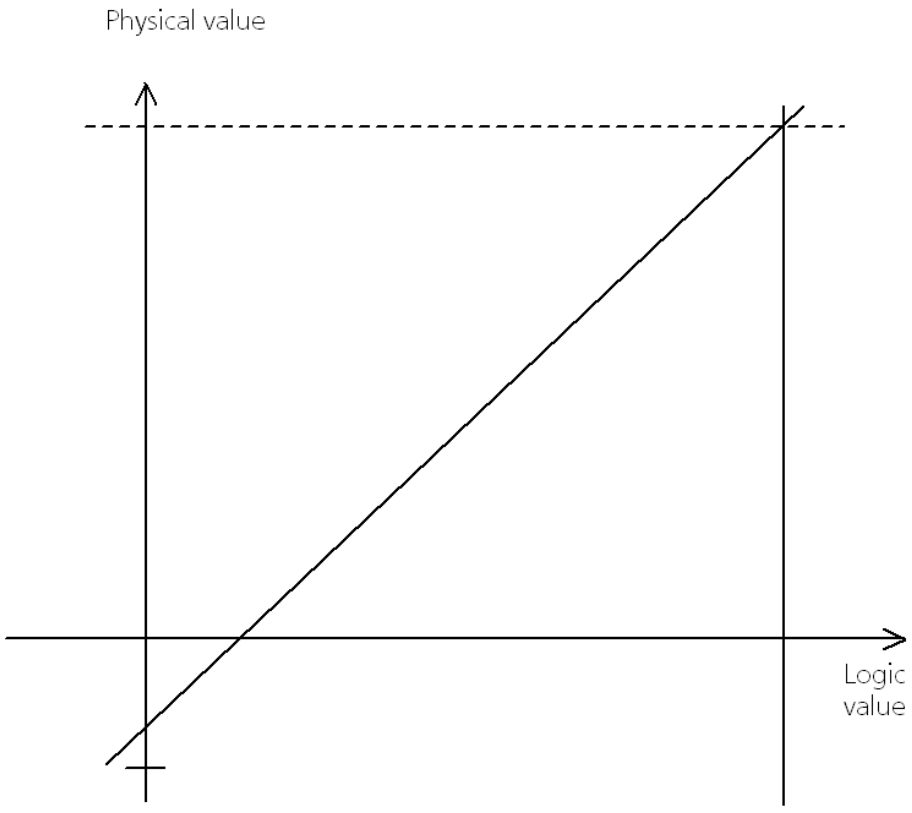


Figure 2.8: Linear Conversion

```
Offset = PhysMin
```

For the values shown in Fig. 2.8 scale and offset evaluates to:

- Scale = 150 °C / 4095 units = 0,03663 °C / unit
- Offset = -30 °C

Chapter 3

Database Editor Operation

3.1 Overview

With the Database Editor you are able to edit the interpretation database for the Data Interpreter Module.
Generally, for all input fields which require integral values, the data can be entered decimally or hex decimally with prefixed "0x".

3.2 Main window

The main window of the Database Editor basically consists of two parts. On the left, a tree view of the current interpretation database is shown. On the right, the information on the currently selected tree-node can be viewed and edited. According to the node type selected (data record type), the form shown on the right changes.

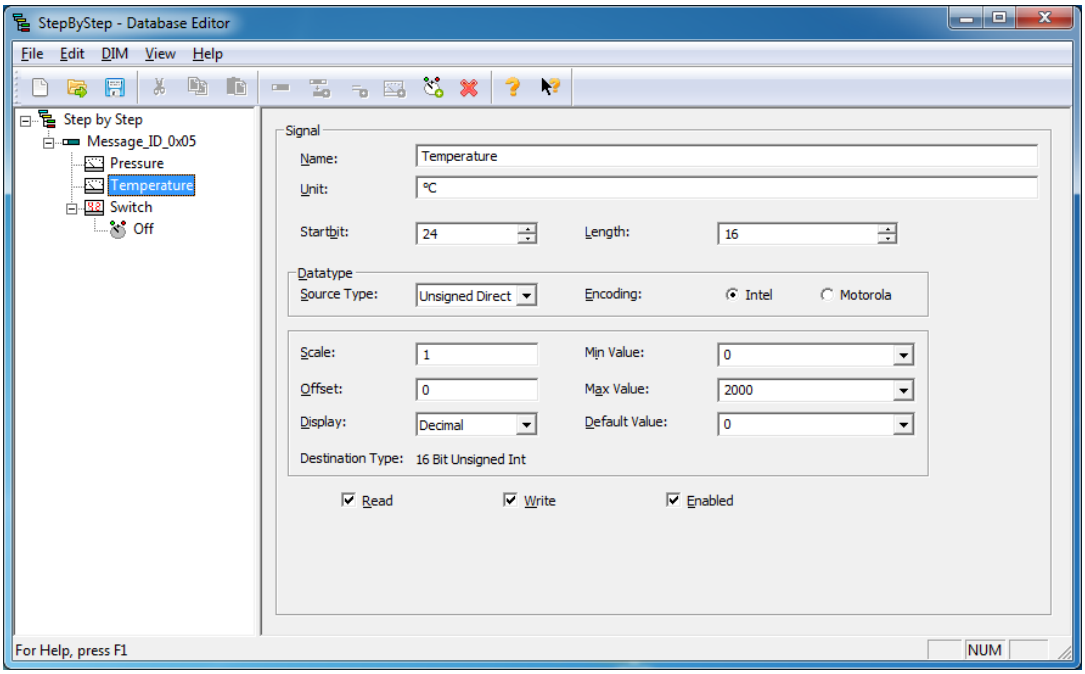


Figure 3.1: The main window of the Database Editor

3.3 Data records in the Data Interpreter database

There are seven different data records in a Data Interpreter database:

- Project
- Message
- Mux masks (bit position/description of a multiplexer)
- Mux values (values belonging to the multiplexers)
- Else values (alternative branch)
- Signal definition (analogue signals)
- States (description of state of signal values)

3.3.1 Project

The project data record is situated at root level. All project specific data are assigned to it. This includes:

Attribute	Comment
Project name	max. 63 characters
Customer	max. 63 characters
User	max. 63 characters
Version	max. 63 characters
Comment	max. 63 characters
Date	max. 63 characters (is updated when saved)
Frame type	11 or 29 bit identifier

One level lower, the following data record-types are permitted:

- none, one or several message objects

The input of the project data is made over the input mask in Fig. 3.2.

3.3.2 Message

In a message definition the message to be interpreted is specified over its CAN identifier and the payload length (data length code, DLC) in bytes. A message object can be seen as a message filter depending on the message identifier and the message length.

Attribute	Comment
Identifier	DWORD
Name	max. 63 characters
Payload	DWORD
Cycletime	resolution in 100 ns

One level lower, the following data record-types are permitted:

- none or several multiplexer independent signals
- at least one MuxMask

The input of the data is made over the input mask in Fig. 3.3.

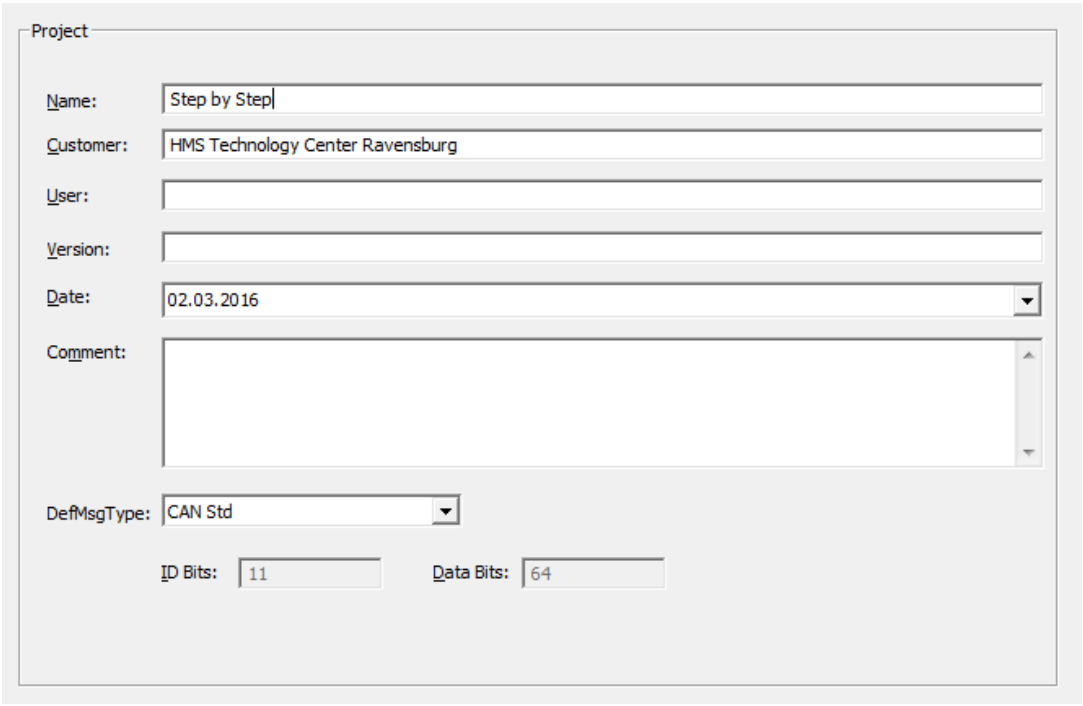


Figure 3.2: Input mask for project data

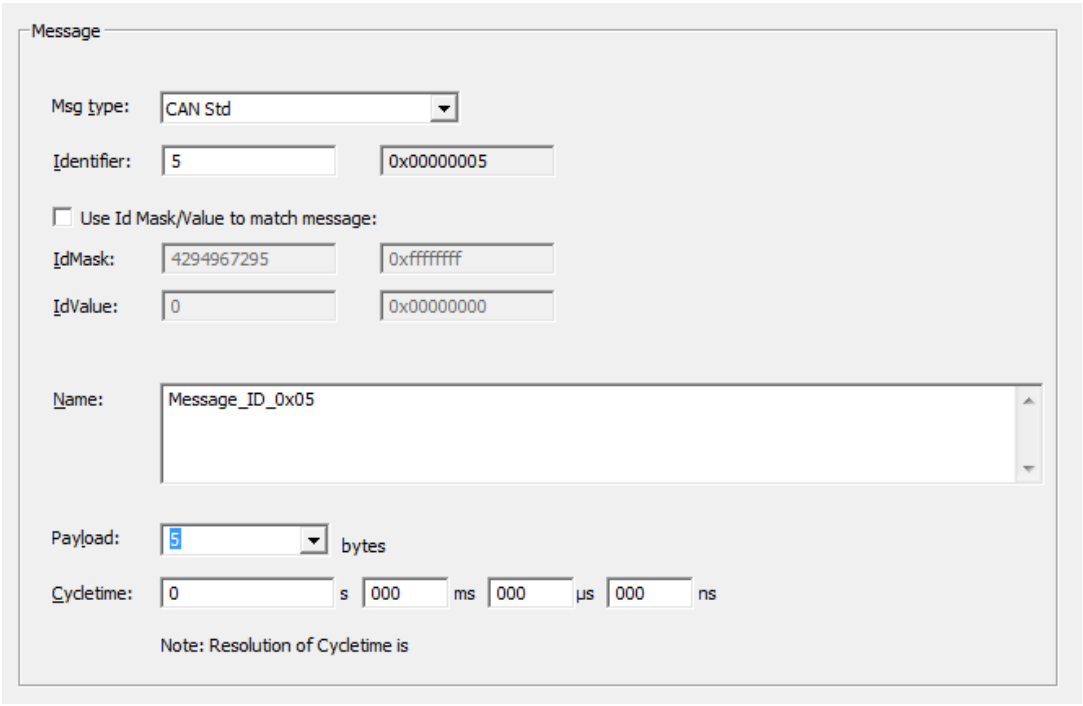


Figure 3.3: Input mask for message data

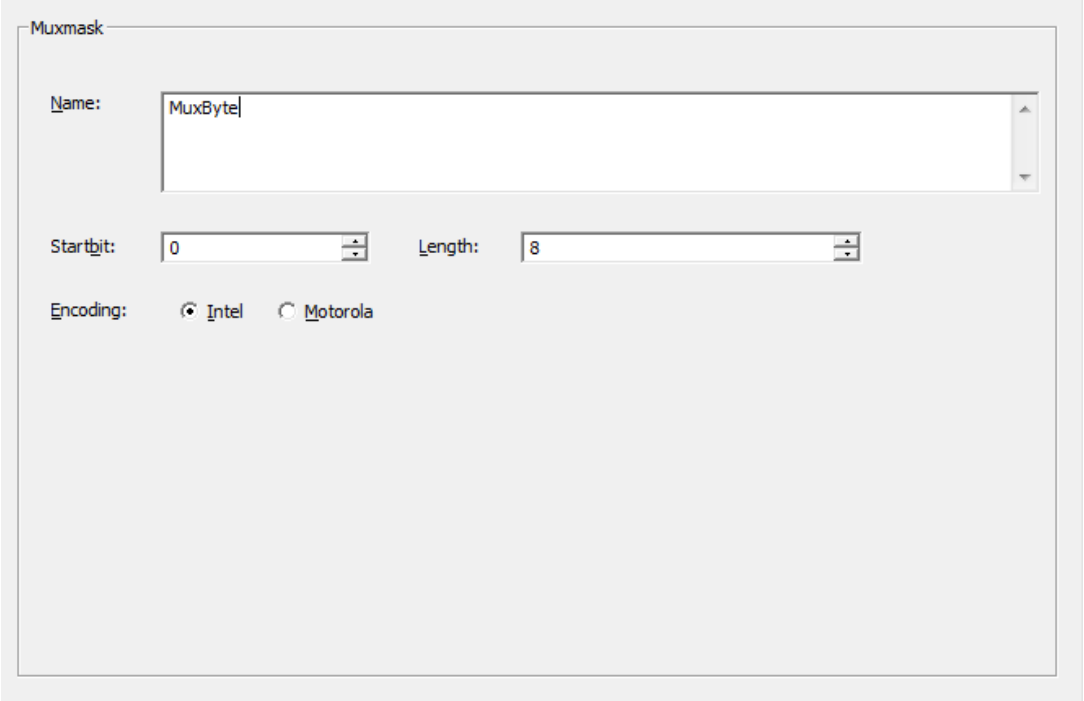


Figure 3.4: Input mask for Mux masks

3.3.3 Mux mask

A Mux mask forms the first part of a multiplexer and determines the area within a CAN telegram with which the remaining data bits are specified.
As a rule, the primary Mux mask is laid over the identifier area of the CAN-telegram. The following attributes are available:

Attribute	Comment
Name	max. 63 characters
Start bit	Integer [0..telegram length-1]
Length	Integer [1..telegram length]

One level lower, the following data record-types are permitted:

- none or several Mux values
- an Else value (alternative branch)

The input of the data is made over the input mask in Fig. 3.4.

3.3.4 Mux value

A Mux value always lies below a Mux mask data record, together with which it forms a multiplexer. It contains the following data:

Attribute	Comment
Name	max. 63 characters
Value	integral

Figure 3.5: Input mask for Mux values

A MuxValue describes the value of a Multiplexers. The attribute "Value" is a comparison value when the data are interpreted.
One level lower, the following data record-types are permissible:

- none or several signal definitions
- maximum one Mux mask

The input of the data is made over the input mask in Fig. 3.5.

3.3.5 Else value

The Else value stands for all Mux values which were not specifically defined. It is thus only a special Mux value which covers the non-specified cases.

Attribute	Comment
Name	max. 63 characters
Status	(not shown by Database Editor)

One level lower, the following data record-types are permitted:

- Signal

ElseValue nodes are coupled to use with Mux masks and are thus deposited automatically in the Database Editor.
The input of the data is made over the input mask in Fig. 3.6.

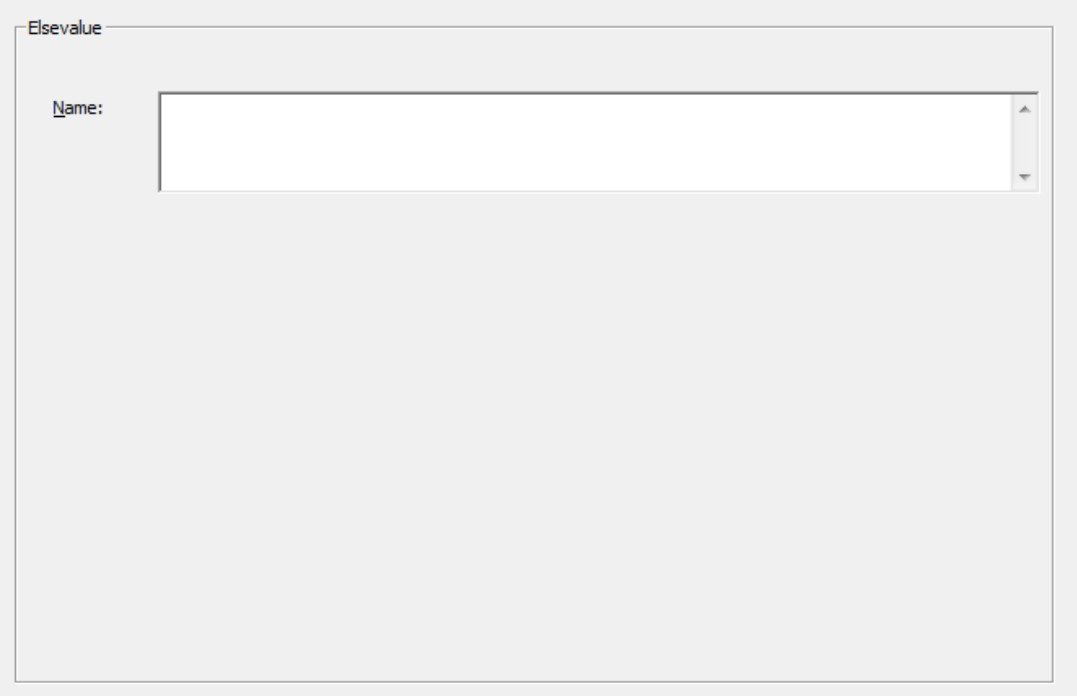


Figure 3.6: Input mask for Else values

3.3.6 Signal definitions

In a signal definition, all data which specifies the structure of a signal is stored.

Attribute	Comment
Name	max. 63 characters
Unit	max. 63 characters
Start bit	Integral [0..telegram length]
Bit length	Integral [0..telegram length]
Scale	Double
Offset	Double
Default value	DIM_VALUE
Min value	DIM_VALUE
Max value	DIM_VALUE
data type	DWORD
Format string	max. 11 characters (not yet supported)
Colour	DWORD (not yet supported)
Status	DWORD (Read, Write, Enabled)

This information is necessary, since the Data Interpreter Module core carries out a translation of signals from a source type into a target type.
The source type of the signals results from the start bit and bit length as well as from the attribute 'Signal type'. This attribute is coded according to bits. The meaning of the individual bits is given in the appendix. Since the Database Editor decodes the coding, it is not relevant at this point. (The three input fields Conversion, Display and Encoding are deposited there).
The attributes 'Scaling' and 'Offset' are used for a linear conversion of the signals.
The attributes "Default value", "Min value" and "Max value" are of the type DIM_VALUE, i.e. their meaning or their value depends on the source type of the signals.
The attributes "FormatString" and "Colour" are planned for future versions but are not currently supported.

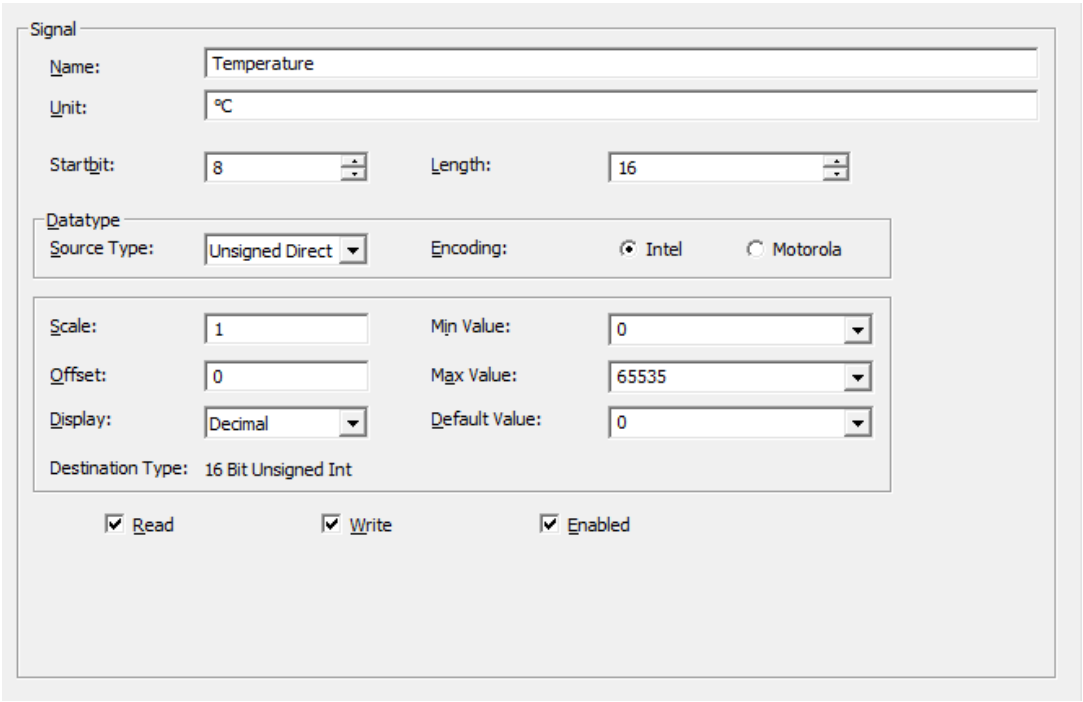


Figure 3.7: Input mask for signal definition

In the "Status" attribute, it is recorded whether a signal is readable, writable or whether it is cleared at all.
One level lower, the following data record-types are permissible:

- none or several states (state descriptions)

The input of the data is made over the input mask in Fig. 3.7.

3.3.7 States

In a state data record, the state which the signal above can accept is specified. These states are described by a name and an assigned value.
States only make sense for signals with an integral value result.

Attribute	Comment
Meaning	max. 63 characters
Value	DWORD

Below a state data record, there can be no further data records.
The input of the data is made over the input mask in Fig. 3.8.

3.4 Menu functions

Fig. 3.9 shows the complete menu of the Database Editor.

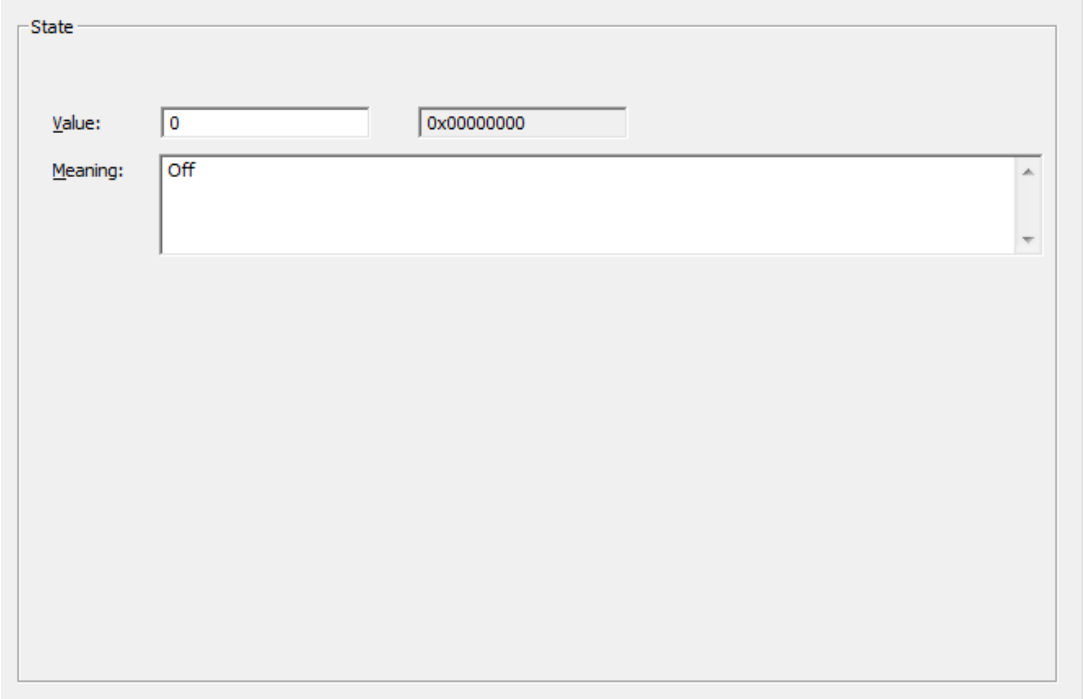


Figure 3.8: Input mask for states

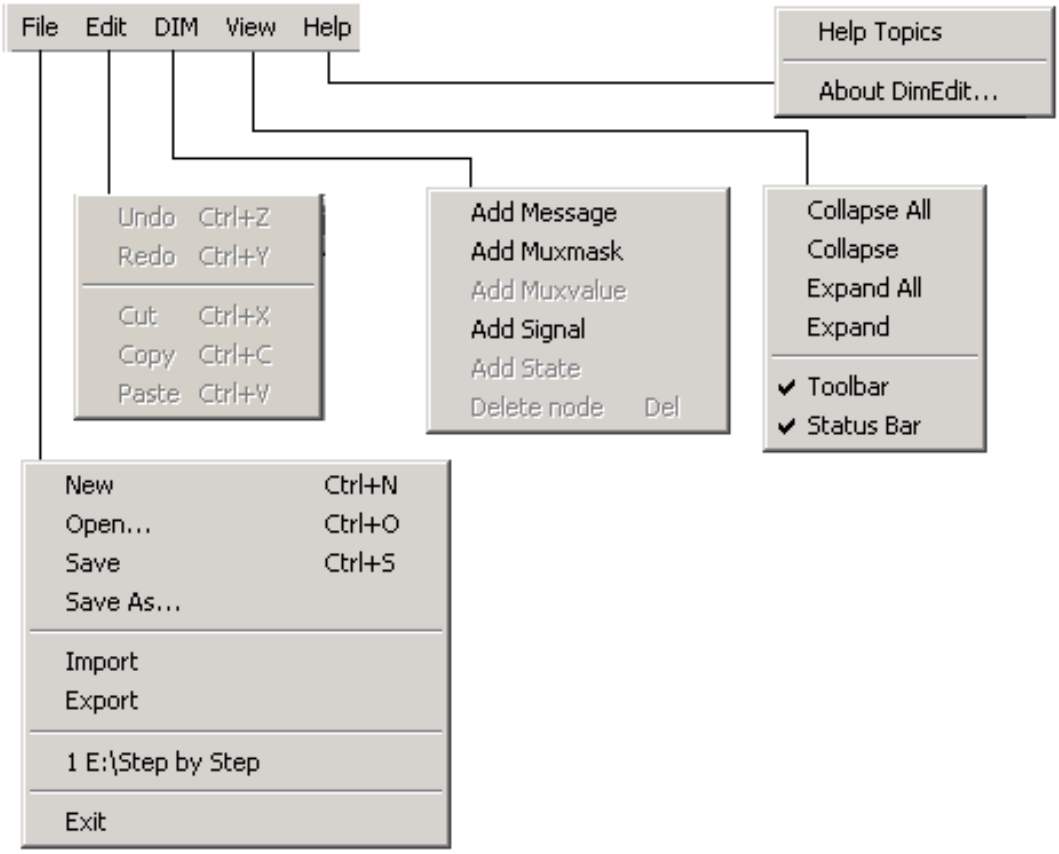


Figure 3.9: Menu functions of the Database Editor

3.4.1 File Menü

The File menu provides the following functions:

Menu point	Function
New	Sets up a new Data Interpreter database (Ctrl-N)
Open	Opens an existing database (Ctrl-O)
Save	Saves the Data Interpreter database under the current name (Ctrl-S)
Save As...	Saves the database under a different name
Import	Imports the Data Interpreter database from an external format
Export	Exports the database to an external format
Recent File List	Contains the files used most recently
Exit	Exits the program

3.4.2 Edit menu

The functions of the Edit menu refer to the active entry field:

Menu point	Function
Undo	Undos a change (Ctrl-Z)
Cut	Deletes selected text (Ctrl-X))
Copy	Copies selected text to clipboard (Ctrl-C)
Paste	Inserts text from clipboard (Ctrl-N)

3.4.3 DIM menu

The DIM-menu provides functions for inserting/deleting Data Interpreter data records:

Menu point	Function
Add Message	Inserts a message object below the selected project node
Add MuxMask	Inserts a multiplexer-mask below the selected data record
Add MuxValue	Inserts a multiplexer value below the selected data record
Add Signal	Inserts a signal definition below the selected data record
Add State	Inserts a state definition below the selected data record
Delete Node	Deletes the current node (Del)

3.4.4 View menu

The View menu provides functions to configure the view.

Menu point	Function
Collapse All	Reduces the tree view to the root node
Collapse	Reduces the marked node
Expand All	Expands all nodes
Expand	Expands all nodes of the marked node
Toolbar	Reveals/conceals the toolbar
Statusbar	Reveals/conceals the status bar

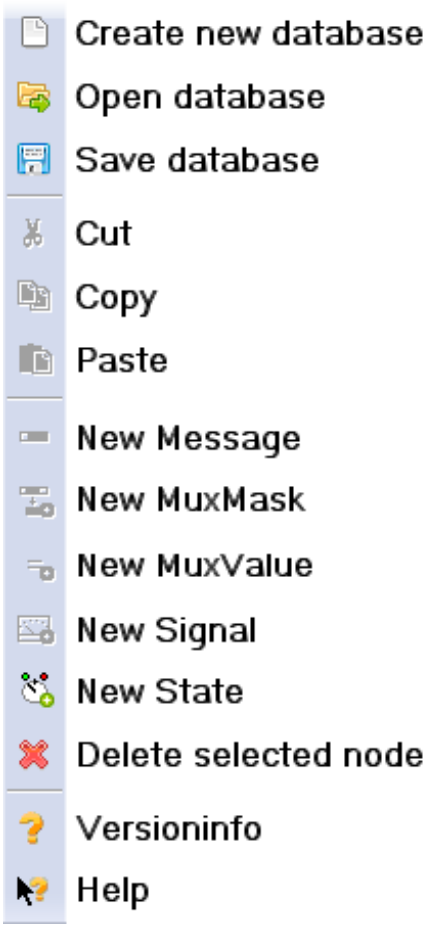


Figure 3.10: Toolbar of Database Editor

3.4.5 Help menu

The Help menu contains functions to call up Help and the About Dialog.

Menu point	Function
Help Topics	Displays the Help file (F1)
About DIMedit	Displays the About dialog

3.5 Toolbar

The toolbar (Fig. 3.10) provides the most important functions of the menu again:

3.6 Clipboard

The clipboard is a software program that is used for short-term storage of data as it is transferred between Database Editor documents or the Database Editor, via copy and paste operations. It is a part of the GUI environment and is implemented as an anonymous, temporary block of memory that can be accessed from the Database Editor within the environment. A selected DIM-node is copied to clipboard by the function "copy" or "cut". By "paste" a node can be inserted in the tree structure. The Clipboard menu provides the following functions:

Menu point	Function
Cut	Copies selected node to clipboard and deletes selected node (Ctrl-X)
Copy	Copies selected node to clipboard (Ctrl-C)
Paste	Inserts node from clipboard (Ctrl-V)

3.6.1 Cut menu point

Cut places a copy of the selected node in the clipboard and removes it from its original location. It is possible to cut all DIM nodes (DIM objects) excluding the project node.

3.6.2 Copy menu point

Copy places a copy of the selected node in the clipboard without removing it from its original location. It is possible to copy all DIM nodes (DIM objects).

3.6.3 Paste menu point

The paste operation inserts visibly the clipboard information at the insertion point. It is possible to paste all DIM nodes (DIM objects) excluding the project node.

3.7 Undo/Redo feature

Undo is a command in Database Editor. It erases the last change done to the DIM-document reverting it back to an older state. The opposite of undo is redo. The redo command reverses the undo or advances the buffer to a more current state.

Menu point	Function
Undo	Erases the last change (Ctrl-Z)
Redo	Reverses the undo (Ctrl-Y)

3.7.1 Undo menu point

It erases the last change done to the DIM-document reverting it back to an older state.

3.7.2 Redo menu point

It reverses the undo or advances the buffer to a more current state.

3.8 Drag and Drop feature

In Database Editor, drag-and-drop is the action of (or support for the action of) clicking on a DIM object and dragging it to a different location or onto another DIM object. In general, it can be used to invoke copy actions of DIM objects.

3.8.1 Select an element

Press, and hold down, the left button on the mouse, to "grab" the object.

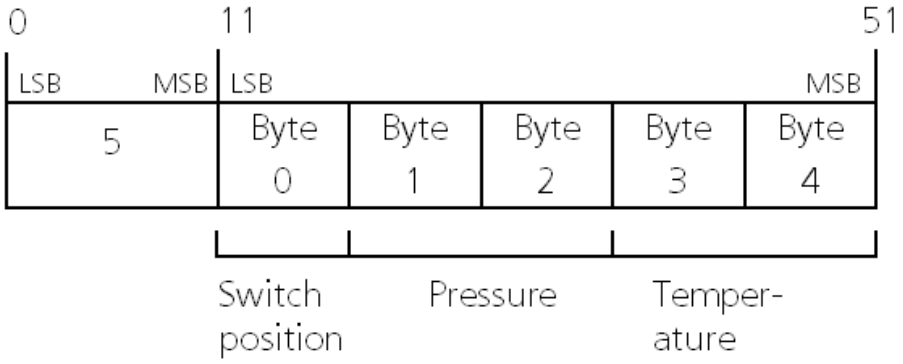


Figure 3.11: Example telegram

3.8.2 Drag

"Drag" the object/cursor/pointing device to the desired location.

3.8.3 Drop

"Drop" the object by releasing the left button.

3.9 Quick Start - Definition of an example database

We now want to draw up a database with the aid of a Database Editor. With this Data Interpreter database we want to describe the protocol of a unit with the following characteristics:

- The unit is fitted with a switch with three positions, a pressure sensor and a temperature sensor
- After switching on the unit, the unit transmits its process signals (switch position, pressure and temperature) cyclically via the CAN bus. The CAN identifier used is ID 5
- The switch position is transmitted in the first data byte and can take on the values 0.1 and 2 for (Off, middle position and full load)
- The next two bytes contain the pressure in hPa in INTEL-Format as DWORD without sign
- In bytes 3 and 4, the temperature is transmitted in °C in INTEL-format as a signed DWORD

The telegram has the layout shown in Fig. 3.11.
Now we want to draw up a project with the aid of the Database Editor, with which this CAN telegram can be interpreted.

3.9.1 Defining a new project

After the Database Editor is started, a blank project is defined. A new project can be started any time with the command **File** | **New**. A blank project contains the project-node only. The project data is displayed on the right and can be altered (Fig. 3.12).

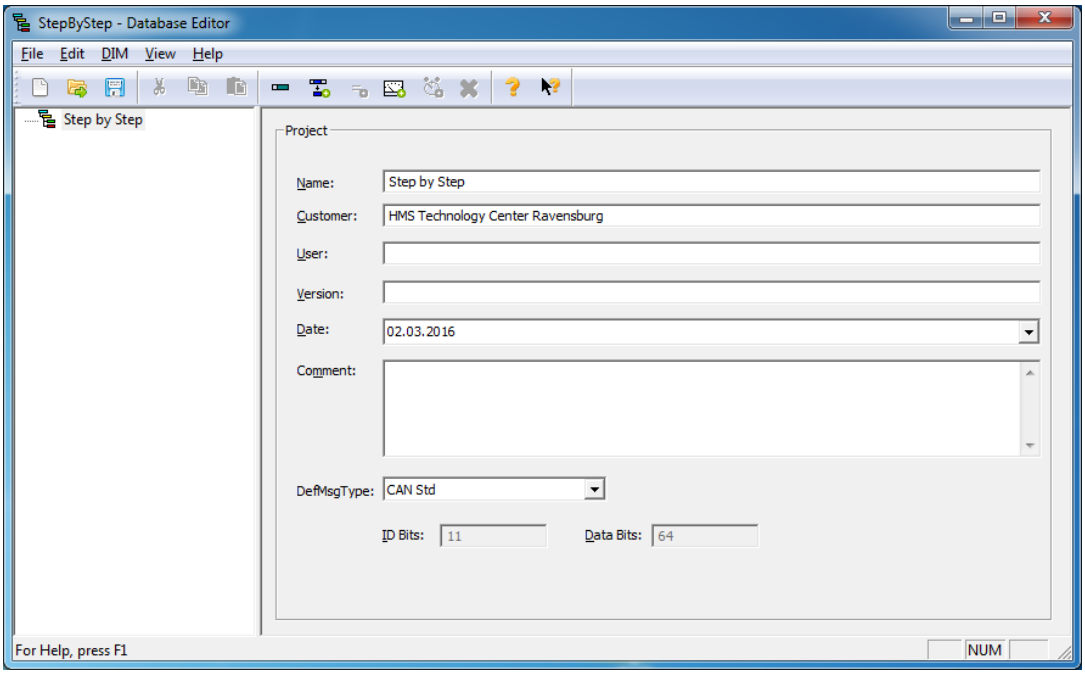


Figure 3.12: Input of project data

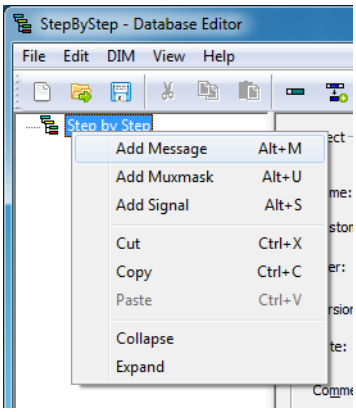


Figure 3.13: Adding a message object

3.9.2 Defining the message object for the CAN-identifier

Only the telegram with the identifier 5 should be interpreted by the Data Interpreter Module. For this reason we define a multiplexer which covers the whole ID area.
For this, click on the project node with the right mouse button. A popup menu appears (Fig. 3.13). Now add a message object by selecting the menu point **Add Message**.
After selecting the newly drawn up node, you obtain an empty input mask. Now define the identifier for the message to be interpreted with "5". Additionally you can assign a name to the message (Fig. 3.14).

3.9.3 Defining signals

After you have defined the message object for the CAN identifier, you have to define the signals contained in the CAN telegram with ID 5. For this, add a signal definition below the message object. To do this, click on the message object with the right mouse button and add a signal definition by selecting **Add Signal** from the context menu (Fig. 3.15).

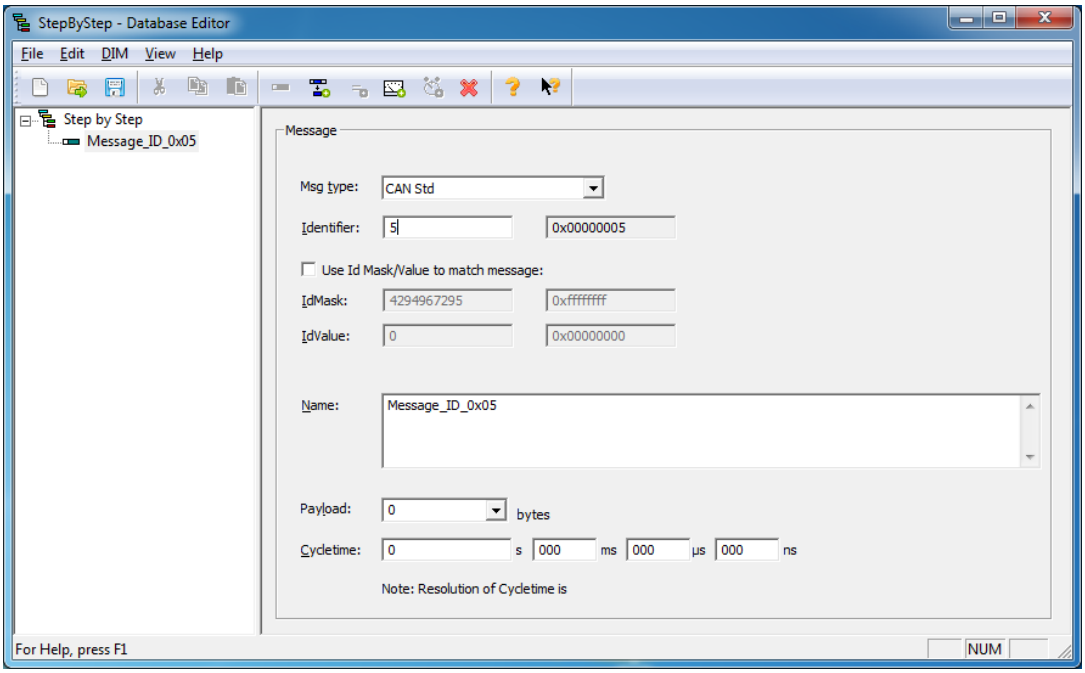


Figure 3.14: Input of message data

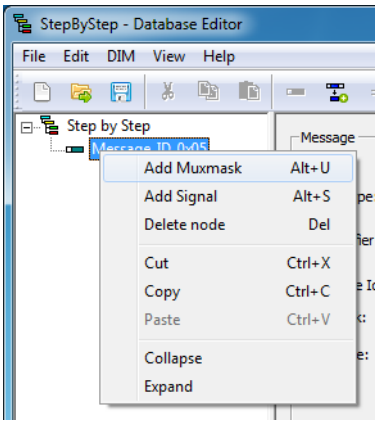


Figure 3.15: Adding a signal definition

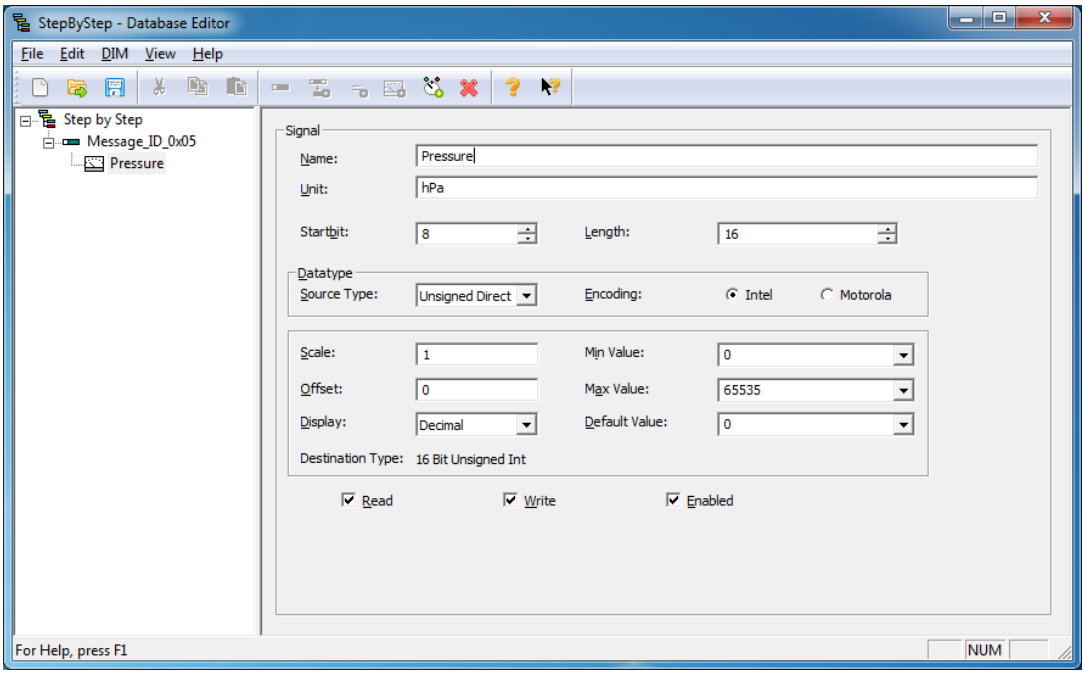


Figure 3.16: Definition of the pressure signal

First determine the settings for the signal "Pressure". According to the default, the pressure information is in byte 2 and byte 3 of the data field. The unit is hPa, the maximum pressure was fixed at 10000 hPa. The value is transmitted directly, therefore Scale is set to 1 and Offset to 0 (Fig. 3.16).
We have set the source data type to a DWORD in INTEL format without sign. Carry out the appropriate settings in the fields Conversion and Encoding. In the Display field, you can parameterise the display by means of the Signal module. Scale and Offset are two parameters of a linear equation with the aid of which a conversion is applied to the signal. The signal values can be checked/preallocated via MinValue, MaxValue and DefaultValue.
Then define the signal "Temperature". It is deposited in byte 3 and 4 of the data field and has a maximum value of 2000 °C. The value is transmitted dircetly, therefore Scale is set to 1 and Offset to 0 (Fig. 3.17).The source data type is in this case a signed DWORD in INTEL format.
Finally define the signal "Switch". The state of the switch is in byte 0 of the data field (Fig. 3.18). The signal Switch is a signal which describes states. The switch has the three positions or states:

- Off (0)
- Middle position (1)
- Full load (2)

3.9.4 Defining states

Now add the definition for the three states to the database. For this, click on the with signal node "Switch" with the right-hand mouse button and obtain the context menu shown in Fig. 3.19. Now add a state-data record by selecting the menu point **Add State**.
After selecting the newly-defined state-data record, you can enter the name and the numeric equivalent of the state (Fig. 3.20).
For the remaining two states "Middle position" and "Full load", proceed in the same way, so that at the end you obtain the project tree shown in Fig. 3.21.
These few steps describe the data in our example CAN telegram with identifier 5.

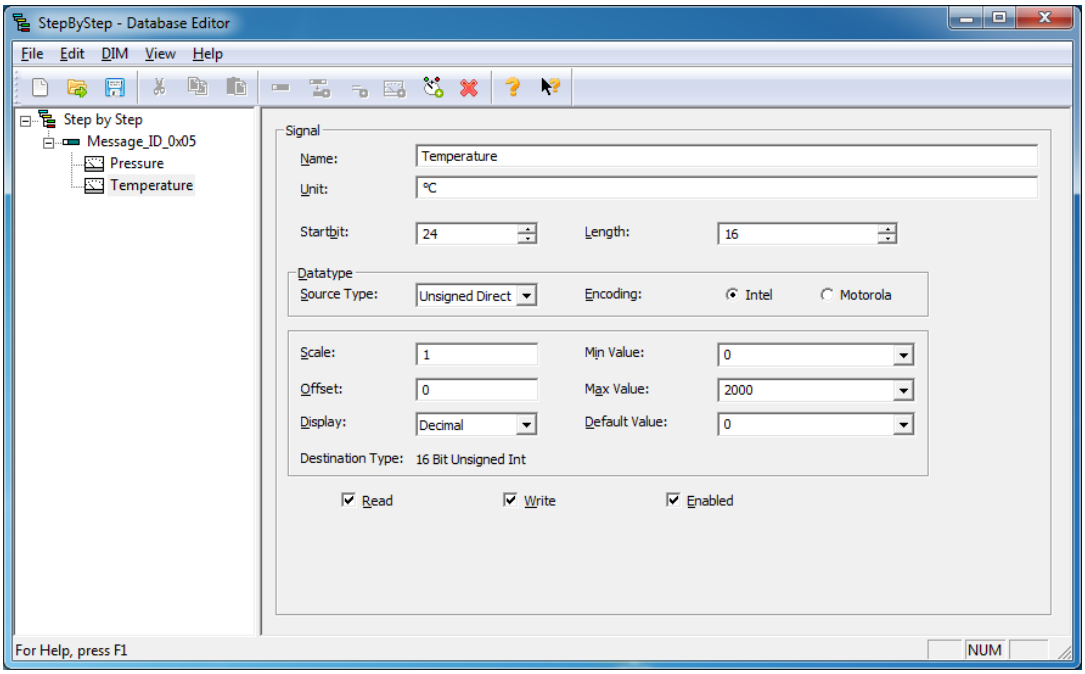


Figure 3.17: Definition of the signal temperature

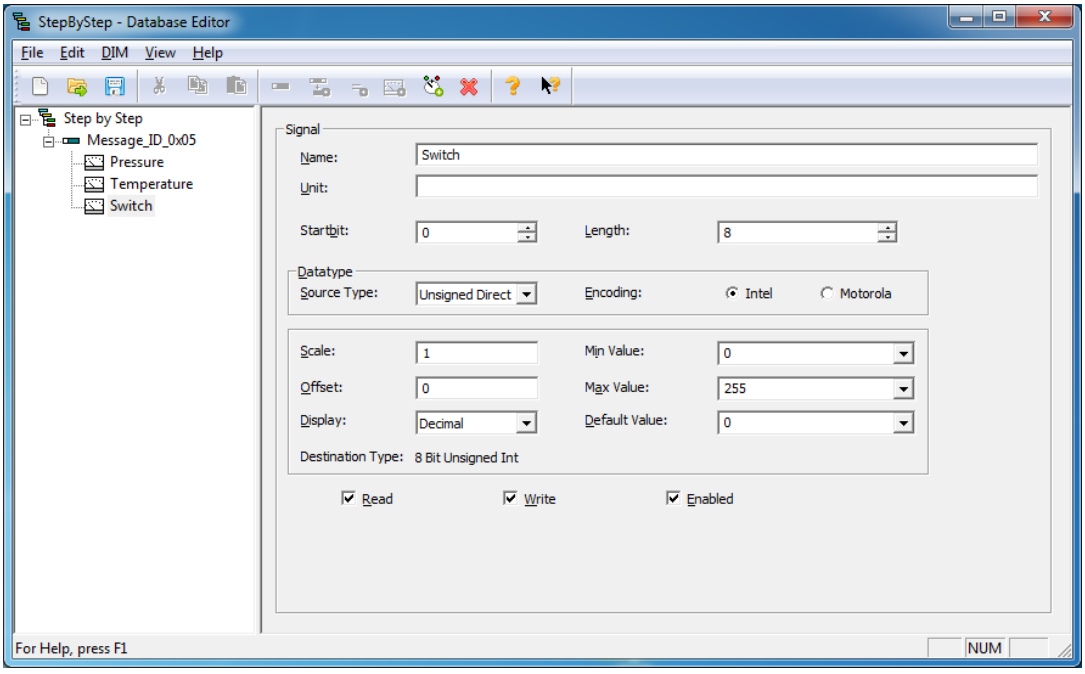


Figure 3.18: Definition of the signal switch

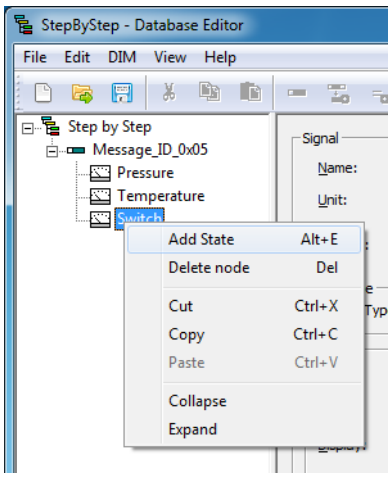


Figure 3.19: Adding a state

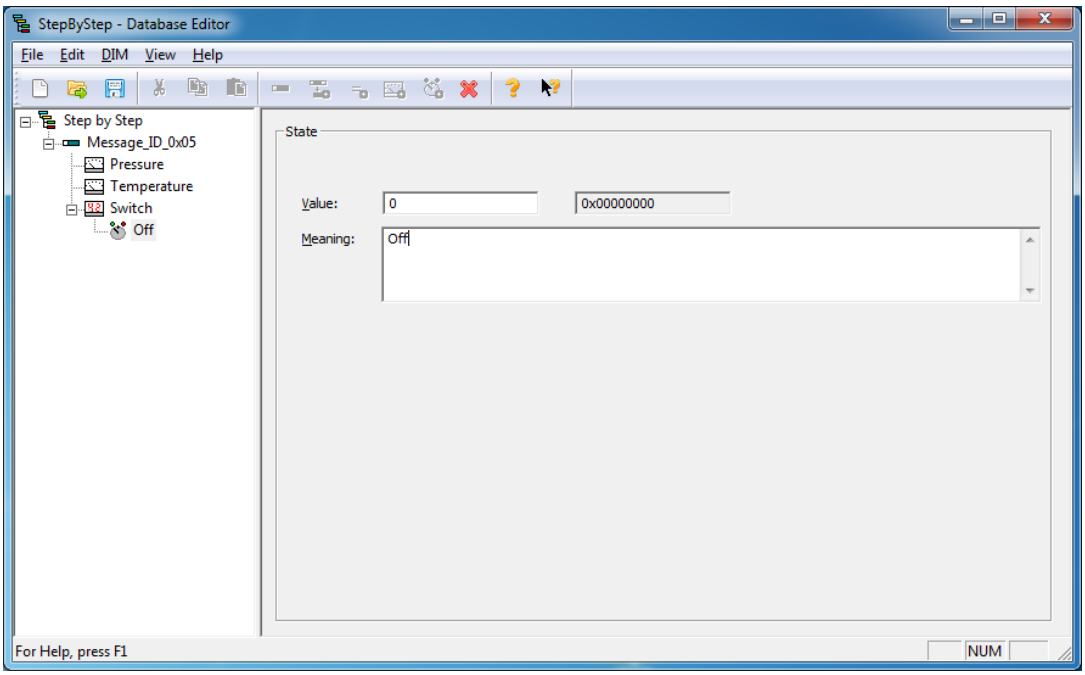


Figure 3.20: Definition of the state "Off"

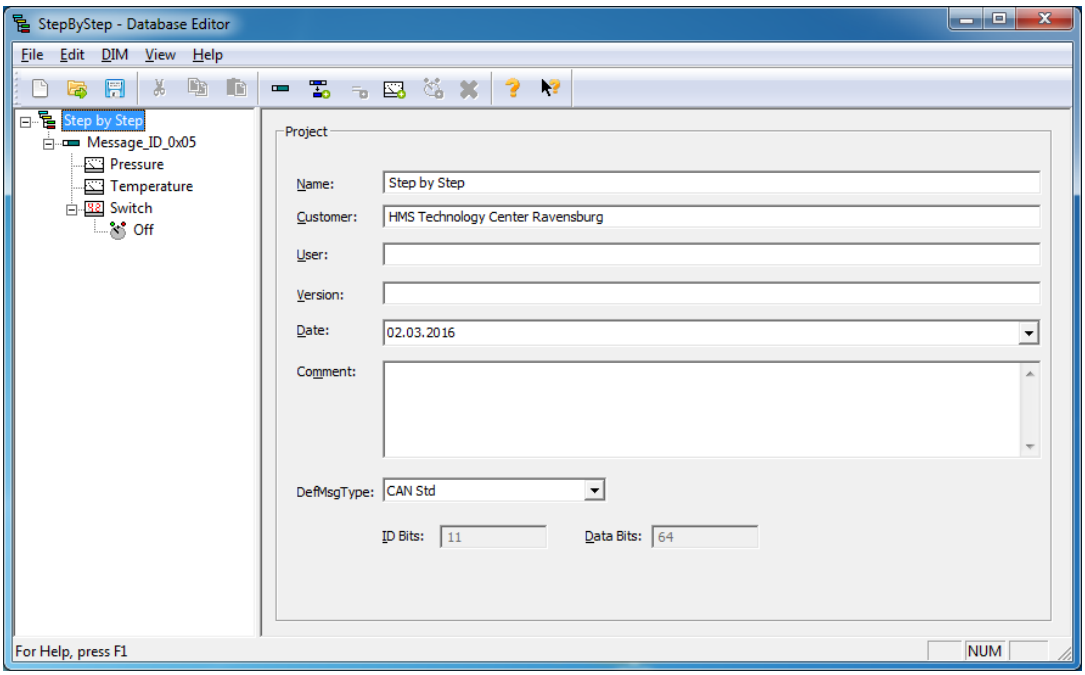


Figure 3.21: The complete project tree

Chapter 4

Available import filters

The following import filters are available:

- own XML format (import and export)
- DCF import (mapped PDOs)
- CANDB import (Vector)

4.1 DCF import (*.dcf)

DCF import is used to extract the PDOs mapped in a CANopen DCF file. This is mainly used to configure OPC servers from IXXAT by means of the CANopen Configuration Studio.

4.2 CANDB import (*.dbc)

When CANDB files are imported, a data interpreter database is generated with the data which are relevant for the interpretation. Further information of CANDB files like environment variables and user defined attributes are ignored by the CANDB import filter.

Chapter 5

Restrictions

- The position of the bits carrying information is described by stating the start bit and bit length. Thus, the Data Interpreter Module cannot process any information which is distributed over unrelated fields in a bit stream.
- States could be defined for all type of variables, but the definition of states is only usable for variables with data type "Unsigned Binary".
- Remote Request Messages are not handled by the Data Interpreter Module. This is not a serious restriction because in RTR Frames no data is contained.

Chapter 6

File format

6.1 Introduction

The project data is stored in a project file which conforms to the XML standard. This appendix describes the elements (tags) used for the structuring of the file and the format of the defined data.

6.2 Structure

6.2.1 Header

Each XML file, and thus also the Data Interpreter project file begins with the following character string:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DIM-PROJECT SYSTEM 'dimprj.dtd'>
```

Here the XML version is given to which the file conforms, as well as the codification of the characters contained. The !DOCTYPE-Tag gives the name of the start tag (DIM-PROJECT) as well as the possible presence of a DTD-file (Document-Type-Definition).

6.2.2 Start tag

The start tag for a Data Interpreter project file is <DIM-PROJECT>. This tag contains all data relevant to a DIM-project.

```
<DIM-PROJECT>

... further data of the DIM-project

</DIM-PROJECT>
```

6.2.3 Project data

Within the <DIM-PROJECT> tag, the project data are summarised with the aid of the tag <PRJ-DATA>. It may only occur once within the project.

```
<DIM-PROJECT>
  <PRJ-DATA>
```

```
    <PRJ-NAME>Demo project</PRJ-NAME>
    <PRJ-CUSTOMER>IXXAT Automation GmbH</PRJ-CUSTOMER>
    <PRJ-USER>U.R.</PRJ-USER>
    <PRJ-VERSION>1</PRJ-VERSION>
    <PRJ-COMMENT>Comment</PRJ-COMMENT>
    <PRJ-DATE>05/06/2004</PRJ-DATE>
    <PRJ-FRAMEFORMAT>0xb</PRJ-FRAMEFORMAT>
    <PRJ-NUMIDBITS>0xb</PRJ-NUMIDBITS>
    <PRJ-NUMDATABITS>0x40</PRJ-NUMDATABITS>
  </PRJ-DATA<

  ... none, one or several messages

</DIM-PROJECT<
```

The individual Tags contain the project data and have the following meaning:

Tag	Meaning
PRJ-NAME	Project name
PRJ-CUSTOMER	Company
PRJ-USER	User
PRJ-VERSION	Version
PRJ-COMMENT	Comment (max. 1024 characters)
PRJ-DATE	Date
PRJ-FRAMEFORMAT	Default frame format used when creating new MESSAGE descriptions
PRJ-NUMIDBITS	number of identifier bits
PRJ-NUMDATABITS>	number of payload bits

For the attribute PRJ-FRAMEFORMAT the following constants are supported:

constant	Meaning
0x02	DIM_FF_CAN11BIT
0x03	DIM_FF_CAN29BIT
0x05	DIM_FF_FLEXRAY
0x06	DIM_FF_CANFD_STD
0x07	DIM_FF_CANFD_EXT

6.2.4 Message data

A message object describes the message to interprete by its CAN identifier. A message definition is made by the tag <Message>. Its data are defined by the tag <MSG-DATA>.

```
<MESSAGE>
  <MSG-DATA>
    <MSG-NAME>MyMessage</MSG-NAME>
    <MSG-LENGTH>0x8</MSG-LENGTH>
    <MSG-ID>0x1a0</MSG-ID>
    <MSG-IDMASK>0xfffffff0</MSG-IDMASK>
    <MSG-IDVALUE>0x200</MSG-IDVALUE>
    <MSG-FRAMEFORMAT>0x2</MSG-FRAMEFORMAT>
    <MSG-CYCLETIME>0</MSG-CYCLETIME>
```

```
      <MSG-STATUS>0x7</MSG-STATUS>
    </MSG-DATA>

    ... maximum one mux maske
    ... one or several signals

  </Message>
```

The individual Tags have the following meaning:

Tag	Meaning
MSG-NAME	Message name
MSG-LENGTH	Message length
MSG-ID	CAN identifier
MSG-FRAMEFORMAT	Message format. The same values as for PRJ-FRAMEFORMAT are allowed.
MSG-CYCLETIME	Cycle time
MSG-STATUS	Status

Later versions from 02/2016 added the following tags to enable message matching by id and mask/value pair:

Tag	Meaning
MSG-IDMASK	id mask
MSG-IDVALUE	id match value

Matching of a message description is done via the following algorithm:

```
if (0xFFFFFFFF == MSG-IDMASK) then
  ismatch = (msg.id == MSG-ID)
else
  ismatch = ((msg.id & MSG-IDMASK) == MSG-IDVALUE)
end
```

6.2.5 Mux mask

A Mux mask definition is included in the tag <MUXMASK>. The data of the Mux mask are marked by the tag <MM-DATA>.

```
<MUXMASK>
  <MM-DATA>
    <MM-NAME>Standard CAN identifier<MM-NAME>
    <MM-STARTBIT>0x15</MM-STARTBIT>
    <MM-BITLENGTH>0xb</MM-BITLENGTH>
  </MM-DATA>

  ... none, one or several mux values
  ... maximum one else value

</MUXMASK>
```

The individual Tags define a Mux mask:

Tag	Meaning
MM-NAME	Name
MM-STARTBIT	Start bit
MM-BITLENGTH	Bit length

6.2.6 Mux value

There can be several Mux value definitions within one mux mask.

```
<MUXMASK>
  <MM-DATA>
    [Daten]
  </MM-DATA>
  <MUXVALUE>
    <VAL-DATA>
      <VAL-NAME>ID_100</VAL-NAME>
      <VAL-VALUE>100</VAL-VALUE>
      <VAL-STATUS>0x7</VAL-STATUS>
    </VAL-DATA>

    ... none, one or several signals

  </MUXVALUE>

... further mux values
... maximum one else value

</MUXMASK>
```

The individual Tags define a Mux value:

Tag	Meaning
VAL-NAME	Name
VAL-VALUE	Value
VAL-STATUS	Status

6.2.7 ElseValue

An Else value represents an alternative branch and thus possesses only the Tags Name and Status.

```
...
<ELSEVALUE>
  <VAL-DATA>
    <VAL-NAME>else</VAL-NAME>
    <VAL-STATUS>0x7</VAL-STATUS>
  </VAL-DATA>

  ... none, one or several signals

</ELSEVALUE>
...
```

The individual Tags have the same meaning as with normal Mux values. The Tag <VAL-VALUE> is ignored within an Else value.

6.2.8 Signal

Signals are the central structure for the Data Interpreter Module. They are included in the tag <VARIABLE>:

```
...
<VARIABLE>
  <VAR-DATA>
    <VAR-NAME>Var_1</VAR-NAME>
    <VAR-STARTBIT>0x20</VAR-STARTBIT>
    <VAR-BITLENGTH>0x8</VAR-BITLENGTH>
    <VAR-UNIT></VAR-UNIT>
    <VAR-DEFVALUE>0</VAR-DEFVALUE>
    <VAR-SCALE>1</VAR-SCALE>
    <VAR-OFFSET>0</VAR-OFFSET>
    <VAR-MINVAL>0</VAR-MINVAL>
    <VAR-MAXVAL>255</VAR-MAXVAL>
    <VAR-STATUS>0x7</VAR-STATUS>
    <VAR-TYPE>0x48010</VAR-TYPE>
    <VAR-DISPCOLOR>0</VAR-DISPCOLOR>
    <VAR-FORMATSTR></VAR-FORMATSTR>
  </VAR-DATA>

  ... none, one or several states

</VARIABLE>
...
```

The following Tags characterise the signal definition:

Tag	Meaning
VAR-NAME	Name
VAR-STARTBIT	Start bi
VAR-BITLENGTH	Bit length
VAR-UNIT	Unit
VAR-DEFVALUE	Default value
VAR-SCALE	Scaling
VAR-OFFSET	Offset
VAR-MINVAL	Minimum value
VAR-MAXVAL	Maximum value
VAR-STATUS	Status
VAR-TYPE	Source type
VAR-DISPCOLOR	Colour
VAR-FORMATSTR	Format string

VAR-TYPE is a 32 bit unsigned value, which specifies the source data type of the signal. Encoding of this value is given in the following table:

Bit	Meaning
0 - 7	Data type:0x10 = Direkt; 0x11 = 1's Complement; 0x12 = 2's Complement; 0x13 = BCD; 0x20 = IEEE; 0x30 = String
8 - 14	reserved
15	Encoding:0x0 = Intel; 0x1 = Motorola
16 - 23	Display Format:0x00 = Format string; 0x01 = Ascii; 0x02 = Binary; 0x03 = Octal; 0x04 = Decimal; 0x05 = Hexadecimal
24 - 30	reserved
31	Sign:0x0 = unsigned; 0x1 = signed

6.2.9 State

States are arranged below signal definitions.

```
[...]
<VARIABLE>
  <VAR-DATA>
    ...
  </VAR-DATA>
  <STATE>
    <STATE-DATA>
      <STATE-MEANING>Aus</STATE-MEANING>
      <STATE-VALUE>0</STATE-VALUE>
    </STATE-DATA>
  </STATE>

  ... further states

</VARIABLE>
...
```

The following Tags belong to a state-data record:

Tag	Meaning
STATE-MEANING	Meaning of the state
STATE-VALUE	Value