

DBC File Format Documentation

Version 1.0.5

This specification as released by Vector is intended for the purpose of information only and is provided on an "AS IS" basis only. To the extent admissible by law, Vector disclaims any warranties or liabilities from the use of this specification. The unauthorized use, e.g. copying, displaying or other use of any content from this document is a violation of the law and intellectual property rights. You must not distribute this specification to anyone outside of the department or the project for which your company has received this specification, and in particular the distribution outside your company is expressly inadmissible.

© 2008 Vector Informatik GmbH

All rights reserved.

Document Management

Revision History		
Version	Editor	Description
1.0 of 2007-02-09	Qu	Specification created
1.0.1 of 2007-11-19	Qu	Description of byte order in signal section corrected. Big endian is stored as "0", little endian is stored as "1".
1.0.2 of 2008-09-03	Sps	The value type "string" of an environment variable is alternatively stored together with the access type in current DBC format
1.0.3 of 2009-08-26	Ne	Description of value ranges and extended multiplexor functionality. Adapted "Signal Definitions" section according to extended multiplexing functionality.
1.0.4 of 2010-02-12	Ne	Added definition of DBC-keywords and printable characters. Added explanation for Pseudo-message 'VECTOR__INDEPENDENT_SIG_MSG'. Corrected: Mask for Extended CAN ID. Corrected: Definition for Signal Value Descriptions
1.0.5 of 2010-04-12	Ne	Added: The backslash isn't allowed in 'char_string'. Corrected: Name of a DBC-identifier is always limited to 32 characters.

Table of contents

1 Introduction	1
2 General Definitions	1
3 Structure of the DBC File.....	2
4 Version and New Symbol Specification.....	3
5 Bit Timing Definition	4
6 Node Definitions.....	4
7 Value Table Definitions.....	4
7.1 Value Descriptions (Value Encodings)	4
8 Message Definitions.....	5
8.1 Pseudo-message	5
8.2 Signal Definitions.....	5
8.3 Definition of Message Transmitters	7
8.4 Signal Value Descriptions (Value Encodings).....	7
9 Environment Variable Definitions	7
9.1 Environment Variable Value Descriptions.....	8

10	Signal Type and Signal Group Definitions	8
11	Comment Definitions	8
12	User Defined Attribute Definitions	9
	12.1 Attribute Definitions.....	9
	12.2 Attribute Values	9
13	Extended Multiplexing	9
14	Examples	10

1 Introduction

The DBC file describes the communication of a single CAN network. This information is sufficient to monitor and analyze the network and to simulate nodes not physically available (remaining bus simulation).

The DBC file can also be used to develop the communication software of an electronic control unit which shall be part of the CAN network. The functional behavior of the ECU is not addressed by the DBC file.

2 General Definitions

The following general elements are used in this documentation:

`unsigned_integer`: an unsigned integer

`signed_integer`: a signed integer

`double`: a double precision float number

`Printable character`: One of the characters 0x20 - 0x7E in the ASCII-Code. I.e. the space character (0x20) is also considered as a printable character.

`char_string`: an arbitrary string consisting of any printable characters except double hyphens ('"') and backslashes ('\'). Control characters like Line Feed, Horizontal Tab, etc. are tolerated, but their interpretation depends on the application¹.

`C_identifier`: a valid C_identifier. C_identifiers have to start with an alpha character or an underscore and may further consist of alpha-numeric characters and underscores.
`C_identifier` = (alpha_char | '_') {alpha_num_char | '_'}

`DBC_identifier`: a C_identifier which doesn't represent a DBC-Keyword.

`DBC-Keyword` = 'VERSION' | 'NS_' | 'NS_DESC_' | 'CM_' | 'BA_DEF_' | 'BA_' | 'VAL_' | 'CAT_DEF_' | 'CAT_' | 'FILTER' | 'BA_DEF_DEF_' | 'EV_DATA_' | 'ENVVAR_DATA_' | 'SGTYPE_' | 'SGTYPE_VAL_' | 'BA_DEF_SGTYPE_' | 'BA_SGTYPE_' | 'SIG_TYPE_REF_' | 'VAL_TABLE_' | 'SIG_GROUP_' | 'SIG_VALTYPE_' | 'SIGTYPE_VALTYPE_' | 'BO_TX_BU_' | 'BA_DEF_REL_' | 'BA_REL_' | 'BA_DEF_DEF_REL_' | 'BU_SG_REL_' | 'BU_EV_REL_' | 'BU_BO_REL_' | 'SG_MUL_VAL_' | 'BS_' | 'BU_' | 'BO_' | 'SG_' | 'EV_' | 'VECTOR__INDEPENDENT_SIG_MSG' | 'VECTOR__XXX'

DBC-identifiers used in DBC files may have a length of up to 32 characters.

¹ The CANdb++ editor e.g. interprets the character combination '0x0D 0x0A' as a Microsoft Windows specific newline and displays this information accordingly in the comment dialog page of the objects. In contrast to this behavior in the 'Unit'-property field of signals the newline couldn't be interpreted meaningful.

Hint: The CANdb++ editor allows for signals, messages, nodes and environment variables to use a name of up to 128 characters. The names of these objects and the associated DBC-identifier are only equal, if the name doesn't exceed 32 characters².

Other strings used in DBC files may be of an arbitrary length (but CANdb++ editor e.g. limits the string length dependent on the objects).

The DBC-keywords used in DBC files to identify the type of an object are given in the following table:

DBC-Keyword	Object Type
BU_	Network Node
BO_	Message
SG_	Signal
EV_	Environment Variable
SIG_GROUP_	Signal Group
VAL_TABLE_	Value Table

The syntax is described using the extended BNF notation (Backus-Naur-Format).

Symbol	Meaning
=	A name on the left of the = is defined using the syntax on the right (syntax rule).
;	The semicolon terminates a definition.
	The vertical bar indicates an alternative.
[...]	The definitions within brackets are optional (zero or one occurrence).
{ ... }	The definitions within braces repeated (zero or multiple occurrences)
(...)	Parentheses define grouped elements.
' ... '	Text in hyphens has to appear as defined.
(* ... *)	Comment.

3 Structure of the DBC File

The DBC file format has the following overall structure:

² To ensure the compatibility the CANdb++ editor always stores C-identifiers for object names with a maximum length of 32 characters and stores the long object names in user defined attributes. The following attributes are used to store names of network nodes, messages, signals, and environment variables longer than 32 characters: "SystemNodeLongSymbol", "SystemMessageLongSymbol", "SystemSignalLongSymbol", and "SystemEnvVarLongSymbol".

```
DBC_file =  
    version  
    new_symbols  
    bit_timing                (*obsolete but required*)  
    nodes  
    value_tables  
    messages  
    message_transmitters  
    environment_variables  
    environment_variables_data  
    signal_types  
    comments  
    attribute_definitions  
    sigtype_attr_list  
    attribute_defaults  
    attribute_values  
    value_descriptions  
    category_definitions      (*obsolete*)  
    categories                (*obsolete*)  
    filter                    (*obsolete*)  
    signal_type_refs  
    signal_groups  
    signal_extended_value_type_list  
    extended_multiplexing ;
```

DBC files describing the basic communication of a CAN network include the following sections:

- `Bit_timing`
This section is required but is normally empty.
- `nodes`
This section is required and defines the network nodes.
- `messages`
This section defines the messages and the signals.

The following sections aren't used in normal DBC files. They are defined here for the sake of completeness only:

- `signal_types`
- `sigtype_attr_list`
- `category_definitions`
- `categories`
- `filter`
- `signal_type_refs`
- `signal_extended_value_type_list`

DBC files that describe the CAN communication and don't define any additional data for system or remaining bus simulations don't include environment variables.

4 Version and New Symbol Specification

The DBC files contain a header with the version and the new symbol entries. The version either is empty or is a string used by CANdb editor.

```
version = ['VERSION' ''] { CANdb_version_string } ' ' ];

new_symbols = [ '_NS' ':' ['CM_'] ['BA_DEF_'] ['BA_'] ['VAL_']
  ['CAT_DEF_'] ['CAT_'] ['FILTER_'] ['BA_DEF_DEF_'] ['EV_DATA_']
  ['ENVVAR_DATA_'] ['SGTYPE_'] ['SGTYPE_VAL_'] ['BA_DEF_SGTYPE_']
  ['BA_SGTYPE_'] ['SIG_TYPE_REF_'] ['VAL_TABLE_'] ['SIG_GROUP_']
  ['SIG_VALTYPE_'] ['SIGTYPE_VALTYPE_'] ['BO_TX_BU_']
  ['BA_DEF_REL_'] ['BA_REL_'] ['BA_DEF_DEF_REL_'] ['BU_SG_REL_']
  ['BU_EV_REL_'] ['BU_BO_REL_'] ['SG_MUL_VAL_'] ];
```

5 Bit Timing Definition

The bit timing section defines the baudrate and the settings of the BTR registers of the network. This section is obsolete and not used any more. Nevertheless the keyword 'BS_' must appear in the DBC file.

```
bit_timing = 'BS:' [baudrate ':' BTR1 ',' BTR2 ] ;

baudrate = unsigned_integer ;

BTR1 = unsigned_integer ;

BTR2 = unsigned_integer ;
```

6 Node Definitions

The node section defines the names of all participating nodes. The names defined in this section have to be unique within this section.

```
nodes = 'BU:' {node_name} ;

node_name = DBC_identifier ;
```

7 Value Table Definitions

The value table section defines the global value tables. The value descriptions in value tables define value encodings for signal raw values. In commonly used DBC files the global value tables aren't used, but the value descriptions are defined for each signal independently.

```
value_tables = {value_table} ;

value_table = 'VAL_TABLE_' value_table_name {value_description}
  ';' ;

value_table_name = DBC_identifier ;
```

7.1 Value Descriptions (Value Encodings)

A value description defines a textual description for a single value. This value may either be a signal raw value transferred on the bus or the value of an environment variable in a remaining bus simulation.

```
value_description = unsigned_integer char_string ;
```

8 Message Definitions

The message section defines the names of all frames in the cluster as well as their properties and the signals transferred on the frames.

```
messages = {message} ;

message = BO_ message_id message_name ':' message_size trans-
        mitter {signal} ;

message_id = unsigned_integer ;
```

The message's CAN-ID. The CAN-ID has to be unique within the DBC file. If the most significant bit of the CAN-ID is set, the ID is an extended CAN ID. The extended CAN ID can be determined by masking out the most significant bit with the mask 0x7FFFFFFF.

```
message_name = DBC_identifier ;
```

The names defined in this section have to be unique within the set of messages.

```
message_size = unsigned_integer ;
```

The message_size specifies the size of the message in bytes.

```
transmitter = node_name | 'Vector__XXX' ;
```

The transmitter name specifies the name of the node transmitting the message. The sender name has to be defined in the set of node names in the node section. If the message shall have no sender, the string 'Vector__XXX' has to be given here.

8.1 Pseudo-message

A pseudo-message with the name 'VECTOR__INDEPENDENT_SIG_MSG' may exist in DBC-files. This message is a DBC-internal construct to store signals which are not mapped on messages.

8.2 Signal Definitions

The message's signal section lists all signals placed on the message, their position in the message's data field and their properties.

```
signal = 'SG_' signal_name multiplexer_indicator ':' start_bit '|'
        signal_size '@' byte_order value_type '(' factor ',' offset ')'
        '[' minimum '|' maximum ']' unit receiver {' ',' receiver'} ;

signal_name = DBC_identifier ;
```

The names defined here have to be unique for the signals of a single message.

```
multiplexer_indicator = ' ' | [m multiplexer_switch_value] [M] ;
```

The multiplexer indicator defines whether the signal is a normal signal, a multiplexer switch for multiplexed signals, or a multiplexed signal. A 'M' (uppercase) character defines the signal as the multiplexer switch. A 'm' (lowercase) character followed by an unsigned integer defines the signal as being multiplexed by the

multiplexer switch. A multiplexed signal is transferred in the message if the switch value of the multiplexer signal is equal to its `multiplexer_switch_value`.

Note: A signal may be a multiplexed signal and a multiplexor switch signal at the same time. And further: more than one signal within a single message can be a multiplexer switch. In both cases the extended multiplexing section (see below) mustn't be empty then.

```
start_bit = unsigned_integer ;
```

The `start_bit` value specifies the position of the signal within the data field of the frame. For signals with byte order Intel (little endian) the position of the least-significant bit is given. For signals with byte order Motorola (big endian) the position of the most significant bit is given. The bits are counted in a sawtooth manner. The startbit has to be in the range of 0 to $(8 * \text{message_size} - 1)$.

```
signal_size = unsigned_integer ;
```

The `signal_size` specifies the size of the signal in bits

```
byte_order = '0' | '1' ; (* 0=big endian, 1=little endian *)
```

The `byte_format` is 0 if the signal's byte order is Motorola (big endian) or 1 if the byte order is Intel (little endian).

```
value_type = '+' | '-' ; (* +=unsigned, -=signed *)
```

The `value_type` defines the signal as being of type unsigned (-) or signed (-).

```
factor = double ;
```

```
offset = double ;
```

The `factor` and `offset` define the linear conversion rule to convert the signals raw value into the signal's physical value and vice versa:

```
physical_value = raw_value * factor + offset
```

```
raw_value = (physical_value - offset) / factor
```

As can be seen in the conversion rule formulas the factor must not be 0.

```
minimum = double ;
```

```
maximum = double ;
```

The `minimum` and `maximum` define the range of valid physical values of the signal.

```
unit = char_string ;
```

```
receiver = node_name | 'Vector__XXX' ;
```

The receiver name specifies the receiver of the signal. The receiver name has to be defined in the set of node names in the node section. If the signal shall have no receiver, the string 'Vector__XXX' has to be given here.

Signals with value types 'float' and 'double' have additional entries in the `signal_valtype_list` section.

```

signal_extended_value_type_list = 'SIG_VALTYPE_' message_id signal_name signal_extended_value_type ';' ;

signal_extended_value_type = '0' | '1' | '2' | '3' ; (* 0=signed or unsigned integer, 1=32-bit IEEE-float, 2=64-bit IEEE-double *)

```

8.3 Definition of Message Transmitters

The message transmitter section enables the definition of multiple transmitter nodes of a single message. This is used to describe communication data for higher-layer protocols. This is not used to define CAN layer-2 communication.

```

message_transmitters = {message_transmitter} ;

Message_transmitter = 'BO_TX_BU_' message_id ':' {transmitter} ';' ;

```

8.4 Signal Value Descriptions (Value Encodings)

Signal value descriptions define encodings for specific signal raw values.

```

value_descriptions = { value_descriptions_for_signal |
                        value_descriptions_for_env_var } ;

value_descriptions_for_signal = 'VAL_' message_id signal_name
                                { value_description } ';' ;

```

9 Environment Variable Definitions

In the environment variables section the environment variables for the usage in system simulation and remaining bus simulation tools are defined.

```

environment_variables = {environment_variable}

environment_variable = 'EV_' env_var_name ':' env_var_type '[' minimum '|' maximum ']' unit initial_value ev_id access_type access_node {',' access_node } ';' ;

env_var_name = DBC_identifier ;

env_var_type = '0' | '1' | '2' ; (* 0=integer, 1=float, 2=string *)

minimum = double ;

maximum = double ;

initial_value = double ;

ev_id = unsigned_integer ; (* obsolete *)

access_type = 'DUMMY_NODE_VECTOR0' | 'DUMMY_NODE_VECTOR1' |
              'DUMMY_NODE_VECTOR2' | 'DUMMY_NODE_VECTOR3' |
              'DUMMY_NODE_VECTOR8000' | 'DUMMY_NODE_VECTOR8001' |
              'DUMMY_NODE_VECTOR8002' | 'DUMMY_NODE_VECTOR8003'; (*
              0=unrestricted, 1=read, 2=write, 3=readWrite, if the value behind 'DUMMY_NODE_VECTOR' is OR-ed with 0x8000, the value type is always string. *)

access_node = node_name | 'VECTOR_XXX' ;

```

The entries in the environment variables data section define the environments listed here as being of the data type "Data". Environment variables of this type can store an arbitrary binary data of the given length. The length is given in bytes.

```
environment_variables_data = environment_variable_data ;  
environment_variable_data = 'ENVVAR_DATA_' env_var_name ':'  
    data_size ';' ;  
data_size = unsigned_integer ;
```

9.1 Environment Variable Value Descriptions

The value descriptions for environment variables provide textual representations of specific values of the variable.

```
value_descriptions_for_env_var = 'VAL_' env_var_name  
    { value_description } ';' ;
```

10 Signal Type and Signal Group Definitions

Signal types are used to define the common properties of several signals. They are normally not used in DBC files.

```
signal_types = {signal_type} ;  
signal_type = 'SGTYPE_' signal_type_name ':' signal_size '@'  
    byte_order value_type '(' factor ',' offset ')' '[' minimum '|' '  
    maximum ']' unit default_value ',' value_table ';' ;  
signal_type_name = DBC_identifier ;  
default_value = double ;  
value_table = value_table_name ;  
  
signal_type_refs = {signal_type_ref} ;  
signal_type_ref = 'SGTYPE_' message_id signal_name ':' sig-  
    nal_type_name ';' ;
```

Signal groups are used to define a group of signals within a messages, e.g. to define that the signals of a group have to be updated in common.

```
signal_groups = 'SIG_GROUP_' message_id signal_group_name repeti-  
    tions ':' { signal_name } ';' ;  
signal_group_name = DBC_identifier ;  
repetitions = unsigned_integer ;
```

11 Comment Definitions

The comment section contains the object comments. For each object having a comment, an entry with the object's type identification is defined in this section.

```
comments = {comment} ;
```

```
comment = 'CM_' (char_string |  
    'BU_' node_name char_string |  
    'BO_' message_id char_string |  
    'SG_' message_id signal_name char_string |  
    'EV_' env_var_name char_string)  
';';
```

12 User Defined Attribute Definitions

User defined attributes are a means to extend the object properties of the DBC file. These additional attributes have to be defined using an attribute definition with an attribute default value. For each object having a value defined for the attribute an attribute value entry has to be defined. If no attribute value entry is defined for an object the value of the object's attribute is the attribute's default.

12.1 Attribute Definitions

```
attribute_definitions = { attribute_definition } ;  
  
attribute_definition = 'BA_DEF_' object_type attribute_name attribute_value_type ';' ;  
  
object_type = '' | 'BU_' | 'BO_' | 'SG_' | 'EV_' ;  
  
attribute_name = '' DBC_identifier '' ;  
  
attribute_value_type = 'INT' signed_integer signed_integer |  
    'HEX' signed_integer signed_integer |  
    'FLOAT' double double |  
    'STRING' |  
    'ENUM' [char_string {',' char_string}]  
  
  
attribute_defaults = { attribute_default } ;  
  
attribute_default = 'BA_DEF_DEF_' attribute_name attribute_value  
    ';' ;  
  
attribute_value = unsigned_integer | signed_integer | double |  
    char_string ;
```

12.2 Attribute Values

```
attribute_values = { attribute_value_for_object } ;  
  
attribute_value_for_object = 'BA_' attribute_name (attribute_value |  
    'BU_' node_name attribute_value |  
    'BO_' message_id attribute_value |  
    'SG_' message_id signal_name attribute_value |  
    'EV_' env_var_name attribute_value)  
    ';' ;
```

13 Extended Multiplexing

Extended multiplexing allows defining more than one multiplexer switch within one message. Further it allows the usage of more than one multiplexer switch value for each multiplexed signal.

The extended multiplexing section contains multiplexed signals for which following conditions were fulfilled:

- The multiplexed signal is multiplexed by more than one multiplexer switch value
- The multiplexed signal belongs to a message which contains more than one multiplexor switch

```
extended multiplexing = {multiplexed signal} ;

multiplexed signal = SG_MUL_VAL_ message_id multiplexed_signal_name
                    multiplexor_switch_name multiplexor_value_ranges ';' ;

message_id = unsigned_integer ;

multiplexed_signal_name = DBC_identifier ;

multiplexor_switch_name = DBC_identifier ;

multiplexor_value_ranges = {multiplexor_value_range} ;

multiplexor_value_range = unsigned_integer '-' unsigned_integer ;
```

Sample:

```
BO_ 100 MuxMsg: 1 Vector_XXX
SG_ Mux_4 m2 : 6|2@1+ (1,0) [0|0] "" Vector_XXX
SG_ Mux_3 m3M : 4|2@1+ (1,0) [0|0] "" Vector_XXX
SG_ Mux_2 m3M : 2|2@1+ (1,0) [0|0] "" Vector_XXX
SG_ Mux_1 M : 0|2@1+ (1,0) [0|0] "" Vector_XXX

SG_MUL_VAL_ 100 Mux_2 Mux_1 3-3, 5-10;
SG_MUL_VAL_ 100 Mux_3 Mux_2 3-3;
SG_MUL_VAL_ 100 Mux_4 Mux_3 2-2;
```

14 Examples

```
VERSION ""
```

```
NS_ :
    NS_DESC_
    CM_
    BA_DEF_
    BA_
    VAL_
    CAT_DEF_
    CAT_
    FILTER
    BA_DEF_DEF_
    EV_DATA_
    ENVVAR_DATA_
    SGTYPE_
    SGTYPE_VAL_
    BA_DEF_SGTYPE_
    BA_SGTYPE_
    SIG_TYPE_REF_
    VAL_TABLE_
    SIG_GROUP_
```

```
SIG_VALTYPE_  
SIGTYPE_VALTYPE_  
BO_TX_BU_  
BA_DEF_REL_  
BA_REL_  
BA_DEF_DEF_REL_  
BU_SG_REL_  
BU_EV_REL_  
BU_BO_REL_  
  
BS_  
  
BU_: Engine Gateway  
  
BO_ 100 EngineData: 8 Engine  
SG_ PetrolLevel : 24|8@1+ (1,0) [0|255] "l" Gateway  
SG_ EngPower : 48|16@1+ (0.01,0) [0|150] "kW" Gateway  
SG_ EngForce : 32|16@1+ (1,0) [0|0] "N" Gateway  
SG_ IdleRunning : 23|1@1+ (1,0) [0|0] "" Gateway  
SG_ EngTemp : 16|7@1+ (2,-50) [-50|150] "degC" Gateway  
SG_ EngSpeed : 0|16@1+ (1,0) [0|8000] "rpm" Gateway  
  
CM_ "CAN communication matrix for power train electronics  
*****  
  
implemented: turn lights, warning lights, windows";  
  
VAL_ 100 IdleRunning 0 "Running" 1 "Idle" ;
```