DBC File Format Documentation Version 01/2007

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


char_string: an arbitrary string consisting of any
     printable characters except double hyphens ('"').


C_identifier: a valid C_identifier. C_identifiers have to
     start witham alpha character or an underscore
     and may further consist of alpha-numeric,
     characters and underscores.


C_identifier = (alpha_char | '_') {alpha_num_char | '_'}
```

C-identifiers used in DBC files may have a length of up to 128 characters. To be compatible to older tools the length should not exceed 32 characters. Other strings used in DBC files may be of an arbitrary length.

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

Keyword	**Object Type**
BU_	Network Node
BO_	Message
SG_	Signal
EV_	Environment Variable

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:

```
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
```

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_'] ];
```

5 Bit Timing Definition

The bit timing section defines the baud rate 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 ]

baud rate  = 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 = C_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 = C_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 = double 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}
```

```
BO_ message_id message_name message = '' message_size transmitter
{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 0xCFFFFFFF.

```
message_name = C_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 massage shall have no sender, the string `'Vector__XXX'` has to be given here.

8.1 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 = C_identifier
```

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

```
multiplexer_indicator = ' ' | 'M' | m multiplexer_switch_value
```

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. Only one signal within a single message can be the multiplexer switch. A 'm' (lowercase) character followed by an unsigned integer defines the signal as being multiplexed by the multiplexer switch. The multiplexed signal is transferred in the message if the switch value of the multiplexer signal is equal to its `multiplexer_switch_value`

```
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 saw-tooth manner.
The start_bit has to be in the range of 0 to (8 * message_size - 1).

```
signal_size = unsigned_integer
```

The signal_size specifies the size of the signal in bits

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

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

8.2 Definition of Message Transmitters

The message transmitter section enables the definition of multiple transmitter nodes of a single node. 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 ';'
```

8.3 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 = C_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'  (*
     0=unrestricted, 1=read, 2=write, 3=readWrite *)

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_aname
{ 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 = C_identifier

default_value = double

value_table = value_table_name

signal_type_refs = {signal_type_ref}

signal_type_ref = 'SGTYPE_' message_id signal_name ':'
     signal_type_name ';'
```

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

```
signal_groups = 'SIG_GROUP_' message_id signal_group_name
     repetitions ':' { signal_name } ';'

signal_group_name = C_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 = '"' C_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 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_ EngineData 100: 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 0 Idle Running "Running" 1 "Idle"
```