## Explanations about the Vector CAN database

***Overview***

The Vector CAN database (*.dbc), that should come together with this documentation, is intended to be used to implement power supplies or electronic loads into Vector company software like CANalyzer or CANoe. This database defines the most important commands available for a particular device, plus necessary calculation factors.

***Limitations***

Due to the structure of the database, not all available commands can be addressed to signals. However, the documentation of the interface hardware (here: CAN interface IF-AB-CAN) includes so-called register lists which list all available commands for a certain device series. The remaining commands can be implemented into the database by the user and used via CAPL.

***What's necessary?***

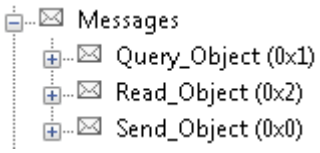
A device with CAN interface module IF-AB-CAN (not IF-AB-CANO), a dedicated database (for example ELR9080-170.dbc), Vector software or compatible software, CAN controller hardware.

***What setup is required on the device side?***

The user has to adjust the CAN base ID (see device and/or interface card documentation) on the device, along with other typical CAN communication settings. When using multiple units on the same bus, the CAN base ID of every unit has to be different.

***What setup is required on the PC side?***

The user has to open the database editor CANdb++ in order to adjust the CAN IDs of the basic messages so they match the device IDs. Every device gets three CAN IDs, based on the adjustable CAN base ID, which is 0x0 by default or after resetting the device to factory settings. The default IDs in the database are thus 0, 1 and 2.




Assignment:
**Query_Object** = Base ID + 1
**Read_Object** = Base ID + 2
**Send_Object** = Base ID,
where the base ID is the CAN base ID as adjusted on the device.
Example: the device has been set to base ID 500. Then the three messages in the database have to be assigned the IDs:
**Query_Object (501), Send_Object (500), Read_Object (502)**

There is a certain number of other messages for cyclic data communication (also see manual of your device, section about CAN interface settings on the control panel). More details below.

***Object definition and compatibility***

The CAN message data is derived from ModBus RTU format. With ModBus, data objects are called registers of 16 bit size. Hence: object number = register number. The device is delivered with documentation, amongst it a programming guide including those register lists. The list, the programming guide and the DBC are the reference for the communication with the device. The CAN interface is supported by following series:

- ELR 9000
- PSI 9000 (alle)
- EL 9000 B
- PSE 9000

***How to use the three basic messages?***

These three messages handle the complete non-cyclic communication with the device.

*Note: sending status or values to the device is considered as remote control and requires to switch the device into remote control condition prior to sending such commands.*

*Use of **Send_Object***

| Name | Send_Object |
|---|---|
| ID | Basis ID |
| Purpose | Sending set values (U, I, P etc.) and status (input/ output on/off etc.)<br>Related ModBus functions: WRITE Single Coil, WRITE Single Register, WRITE Multiple Registers |
| Length | 4-8 Bytes (register number plus argument) |

Send_Object is used to send set values or device conditions to the device. It is required to a) select which set value/device condition to set and b) to give the set value/device condition itself. The selection of the set value or device condition is done via signal Mux_Send, which is connected to a symbolic table that enables you to select like „Output voltage“.

*Use of **Query_Object***

| Name | Query_Object |
|---|---|
| ID | Base ID + 1 |
| Purpose | Query actual value and status, but also currently active set values and more<br>Related ModBus functions: READ Holding Registers, READ COILS |
| Length | 2 Bytes (register number) |

This message is used to query information from the device and it only required to tell what information is queried. Selection is done with signal Mux_Query and the related, symbolic table which is self-explaining. The expected answer is received on message “Read_Object” (see below).

Use of **Read_Object**

| Name | Read_Object |
|---|---|
| ID | Base ID + 2 |
| Purpose | This ID is used by the device to send answers to queries or communication errors |
| Length | 3…8 Bytes (register number plus data) |

This message automatically sorts incoming information based on the ID into signals. Usually, every command that is used with **Query_Object** and causes the device to return data on the base ID + 2, is represented in **Read_Object**. The data in the signals can be used for further purposes.

***How to interpret the incoming data from the device?***

- Set values and actual values are calculated into real values by the DBC using translation factors
- Statuses like „Output on/off“ are described in the assigned value tables
- Error codes are described by value tables

***Notes***

- In case a value is read from a device and displayed in a typical Vector panel output box and that box is marked in red while the value is actually not exceeded, this is caused by translation errors. In order to avoid that you just need to lower the last digit of the corresponding calculation factor by 1.
- Every device model requires a dedicated DBC because of the varying translation factors. Messages and signals of the DBC will not vary within a device series, so new DBC for a different model can easily be created by duplication and simply calculating the corresponding value translation factors.
- A device uses several IDs: a base ID (total: 3 IDs), a broadcast ID, another base ID for cyclic read (total: 5 IDs) and yet another base ID for cyclic send (total: 2 IDs)
- The separately adjustable broadcast ID can be used to send one message with set values or status to all unit of a CAN channel where the broadcast ID matches the device setting.
- All IDs of a device must not overlap with those of any other device
- The time between two subsequent messages must not be too short. The programming guide contains timing recommendation and although not particularly for CAN, we recommend to not read from or write to the device faster than every 5 ms.

***Structure of the standard CAN telegrams***

## *Send_Object*

Base ID and broadcast ID: WRITE Single Coil and WRITE Single Register
(6 Bytes, the rest of the data bytes is 0)

| Bytes 0+1 | Byte 2 | Bytes 3+4 |
|---|---|---|
| Register | Number | Data word |
| 0…65534 | 1 | Value to write |

Base ID: WRITE Multiple Registers
Used for data from a register length of 2. The identification 0xFF identifying the first message, 0xFE the second and so on…

| Bytes 0+1 | Byte 2 | Byte 3 | Bytes 4-7 |
|---|---|---|---|
| Start reg. | Number | ID | Data bytes |
| 0…65534 | 2…123 | 0xFF, 0xFE… | n values to write |

## *Query_Object*

Base ID + 1: READ Holding Registers and READ Coils

| Bytes 0+1 |
|---|
| Register |
| 0…65534 |

## *Read_Object*

Base ID + 2: Response message (if number of registers to read is 1-3)

| Bytes 0+1 | Byte 2-3 |
|---|---|
| Register | 2 data bytes |
| 0…65534 | Queried data |

or

| Bytes 0+1 | Byte 2-5 |
|---|---|
| Register | 4 data bytes |
| 0…65534 | Queried data |

or

| Bytes 0+1 | Byte 2-7 |
|---|---|
| Register | 6 data bytes |
| 0…65534 | Queried data |

Base ID + 2: Response message (if number of registers to read is >3)

| Bytes 0+1 | Byte 2 | Byte 3-7 |
|---|---|---|
| Register | ID | 5 data bytes |
| 0…65534 | 0xFF, 0xFE… | Queried data |

Base ID + 2: Error message

| Bytes 0+1 | 2 |
|---|---|
| Register | |
| 65535 | Error code |

***Cyclic data***
***< from KE firmware 2.13 von PSI/ELR 9000 and EL 9000 B >***

Beside the normal host based CAN communication the device can cyclically send specific data to the host using separate IDs. As long as the cyclic communication is activated, the device will send its status, actual values, set values and adjustment limit in an adjustable interval. With the “Base ID Send”, as adjustable on the device, it supports to send the set values of U, I, P and R in one message rather than with single objects.

Recommended message names for data base integration:
***Send_Mode*** = Base ID Send
***Send_Set_Values*** = Base ID Send +1
***Read_Status*** = Base ID Read
***Read_Act_Values*** = Base ID Read + 1
***Read_Set_Values*** = Base ID Read + 2
***Read_Limits_1*** = Base ID Read + 3
***Read_Limits_2*** = Base ID Read + 4

The intervals for cyclic read and send can be adjusted separately from each other on the device, within a range of 20…5000 ms. All these cyclic messages can be deactivated by setting the interval to 0.

***Structure of the cyclic CAN telegrams***

***Send_Modes***

Base ID Send: Control / models
(2 bytes, the rest is 0)

| Bytes 0-1 |
|---|
| Control |
| Control word |

For the layout of the 16 bit value refer to the external document „Programming ModBus & SCPI“, section 8 for CAN.

***Send_Set_Values***

Base ID Send + 1: Set values (Uset, Iset, Pset, Rset)
(8 Bytes)

| Bytes 0-1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|
| Uset | Iset | Pset | Rset |
| Value | Value | Value | Value |

***Read_Status***

Base ID Read: Status
(4 Bytes, the rest is 0)

| Bytes 0+1+2+3 |
|---|
| Status |
| Value |

For the layout of the 16 bit value refer to the external document „Programming ModBus & SCPI“, section 8 for CAN.

### *Read_Actual_Values*

Base ID Read + 1: Actual values (Uactual, Iactual, Pactual)
(6 Bytes, the rest is 0)

| Bytes 0+1 | Bytes 2+3 | Bytes 4+5 |
|---|---|---|
| Uactual | Iactual | Pactual |
| Value | Value | Value |

### *Read_Set_Values*

Base ID Read + 2: Set values (Uset, Iset, Pset, Rset)
(8 Bytes)

| Bytes 0+1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|
| Uset | Iset | Pset | Rset |
| Value | Value | Value | Value |

### *Read_Limits1*

Base ID Read + 3: Adjustment limits 1 (Imax, Imin, Umax, Umin)
(8 Bytes)

| Bytes 0+1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|
| Imax | Imin | Umax | Umin |
| Value | Value | Value | Value |

### *Read_Limits2*

Base ID Read + 4: Adjustment limits 2 (Pmax, Rmax)
(4 Bytes, the rest is 0)

| Bytes 0+1 | Bytes 2+3 |
|---|---|
| Pmax | Rmax |
| Value | Value |