AMK

AMKASYN

VARIABLE SPEED DRIVES

AMKASYN

Digital drive systems

Option card AE-PSC

- **Programmable control PS**
- **CAN Interface (CAN-S)**

Option card for AZ/AW systems in the version AZ-CNS
Option card for KU systems in the version KU-PSC

CAN Network configuration

Important notes

Due to possible destruction of components by static discharge, touching the electrical connections on the option card should be avoided.

Please attach option card directly from the packaging in the option slot of the KU or AZ module without exerting force and secure with the screw on the front panel.

3701.E Part-No.: 28684

AMK Arnold Müller, Antriebs- und Steuerungstechnik GmbH & Co. KG, D-73221 Kirchheim/Teck,
Tel.: 07021/50 05-0, Telefax: 07021/50 05-176, E-Mail: Info@amk-antriebe.de

Inhalt

List of Figures

1 Abbreviations and explanations

AE-PSC	**A**MKASYN **E**xtension **PS C**AN
APROS	AMK PS programming software
Arbitration	Bus access method; method with which access to the bus is regulated. Solution of the conflict if several stations want to send a message at the same time
AZ/AW system	AMKASYN modular drive system, consisting of central module and inverter modules
AZ-CNS	AMKASYN option card for AZ modules
Broadcasting	describes the possibility of addressing all subscribers to the network simultaneously
CAN	**C**ontroller **A**rea **N**etwork
CANconv	AMK Can converter auxiliary program for transferring the CAN network configuration to the master
ccb	**C**AN **c**onfiguration **b**inary file type *.ccb
ccf	**C**AN **c**onfiguration **f**ile *.ccf
CiA	**C**AN **i**n **A**utomation, international users and manufacturers group e.V.
DVLader	AMK auxiliary tool for flash database access
Emergency Service	Bus fault characteristic on failure of one or several subscribers.
Telegram header	Header information of a message (e.g. priority...)
Ident number	(ID No.) Parameter for parameterizing the AMKASYN system
NMT service	Network management service (network initialization, bus error monitoring, status monitoring of the individual devices)
Node Guarding	Network node monitoring, is performed by the NMT master
Parameter	(ID No.) by which the AMKASYN systems are parameterized
KU	AMKASYN digital compact converter
KU-PSC	AMKASYN option card for KU system
Life Guarding	NMT slave monitors whether the network node monitoring of the NMT master is performed.
PDO	**P**rocess **D**ata **O**bject
PS	**P**rogrammable control
R-PDO	**R**eceive **PDO**
SDO	**S**ervice **D**ata **O**bject
T-PDO	**T**ransmit **PDO**

2 CAN BUS Interface

The CAN interface integrated on the AE-PSC option card fulfils the standard CiA CAN 2.0B and extends this.
To satisfy the tasks of drive systems, the AMK CAN interface offers apart from the standard CAN data channel a synchronous clock signal as extension and is designated as CAN-S. Thus apart from the demand data (parameters, commands, diagnosis) in addition synchronous data (setpoints, actual values, real time bit messages) are transmitted exactly synchronized to one another.

The CAN-S interface consists of the CAN data channel and a synchronous clock signal with which all bus subscribers are synchronized exactly to a master clock.

3 CANopen

The CAN communication is based on the CANopen standard CiA Draft Standard 301 Version 4.01, thus further components corresponding to the standard of external manufacturers can be integrated into the BUS system (e.g. I/Os or gateways). The following functionality is supported:

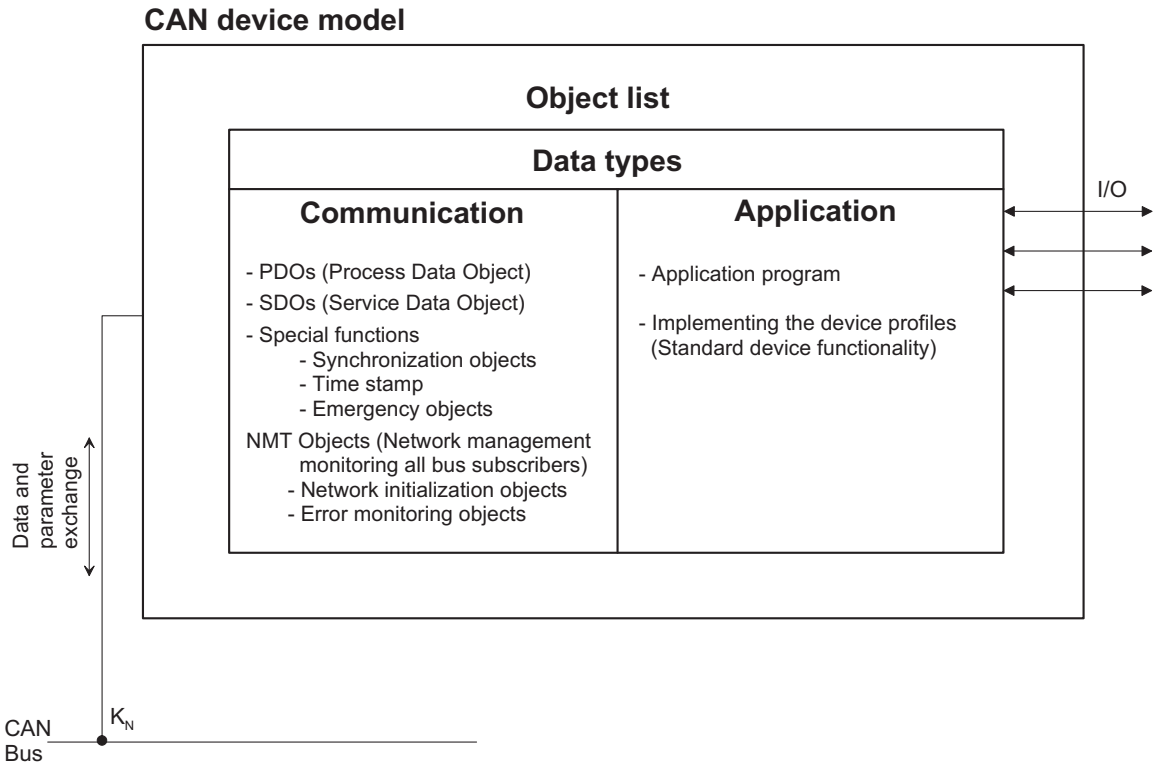
- CANopen Minimum Capability Device Boot-Up
- Node Guarding / Life Guarding
- Transmit PDOs
- Receive PDOs
- Client/Server SDOs
- Emergency Object
- Synchronization Object

3.1 Object list

Each CANopen device has a CANopen object list. The object list is divided into different areas. There are areas for the description of the data types, of the communication and of the application. All data which can be exchanged through the CAN network are represented by corresponding objects in the object list. Access to entries of the object list is made through a 16-bit index and an 8-bit subindex.

The object list is the data and parameter interface of the node to the CAN network. It describes the device with regard to its application and communication properties.

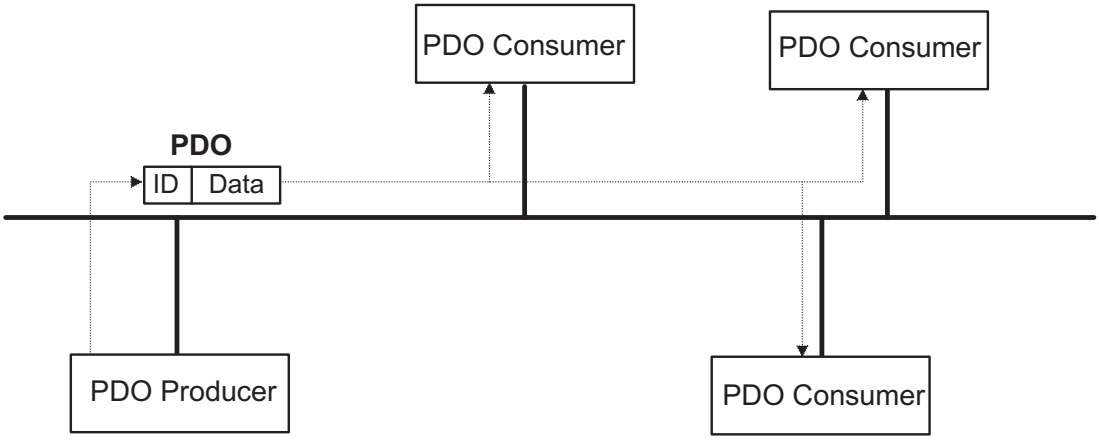
Figure 3-1 CANopen communication model



3.2 Real time communication

So-called PDOs (Process Data Objects) are used for exchanging real time data. PDO communication can be described with the producer / consumer model. The process data are transmitted by a node (producer) and received by one or several nodes (consumers). PDOs are not confirmed by the receiver. In PDOs all maximum 8 data bytes of a CAN frame are available for the data exchange.

Figure 3-2 Process Data Object PDO



3.3 Communication profile

The communication profile is the part of the object list which determines the communication properties of a node (see object list figure). Therefore the communication profile of the object list contains entries which describe the properties of PDOs.

3.4 Synchronous and Asynchronous PDO transmission

The following PDO types are distinguished.

-Synchronous (CycSync n)
-Asynchronous (AMKevent)

Synchronous transmission of a message means that the transmission of a message is fixed in time respect to the transmission of the SYNC message (SYNC object). The synchronisation object is broadcasted periodically by the SYNC producer every ID2 SERCOS Cycle time. This SYNC provides the basic network clock and defines the communication cycle periode. The synchronisation object carries no data and is easy to generate. After SYNC object synchronous messages will be send within the synchronous window length. Asynchronous PDOs will be send after Synchronous. Asynchronous TPDOs are transmitted without any relation to SYNC. The data of asynchronus RPDOs is passed directly to the application.

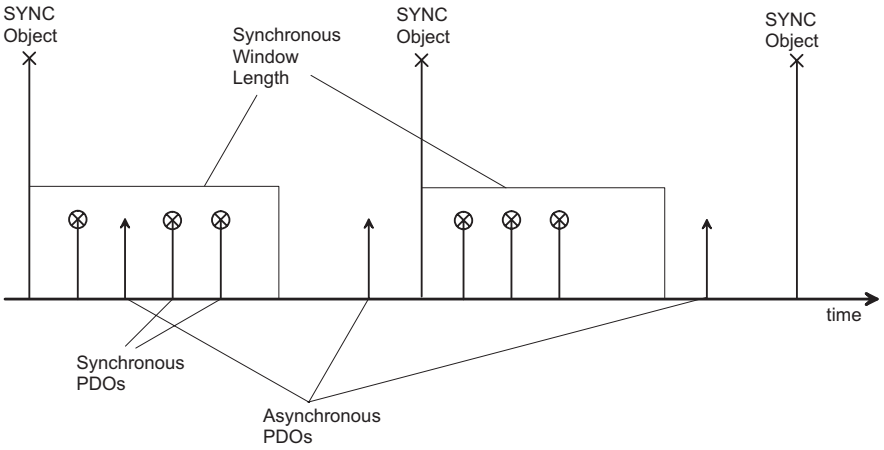


Figure 3-3: Synchronous and Asynchronous PDO transmission

The jitter of this SYNC depends on the bit rate of the bus as even the very high priority SYNC has to wait for the current message on the bus to be transmitted before it gains bus access.

Received synchronous messages will be actuate after the next SYNC object. After actuation the synchronous messages become active in the slaves. The following figure shows the prinziple of SYNC PDOs transmission

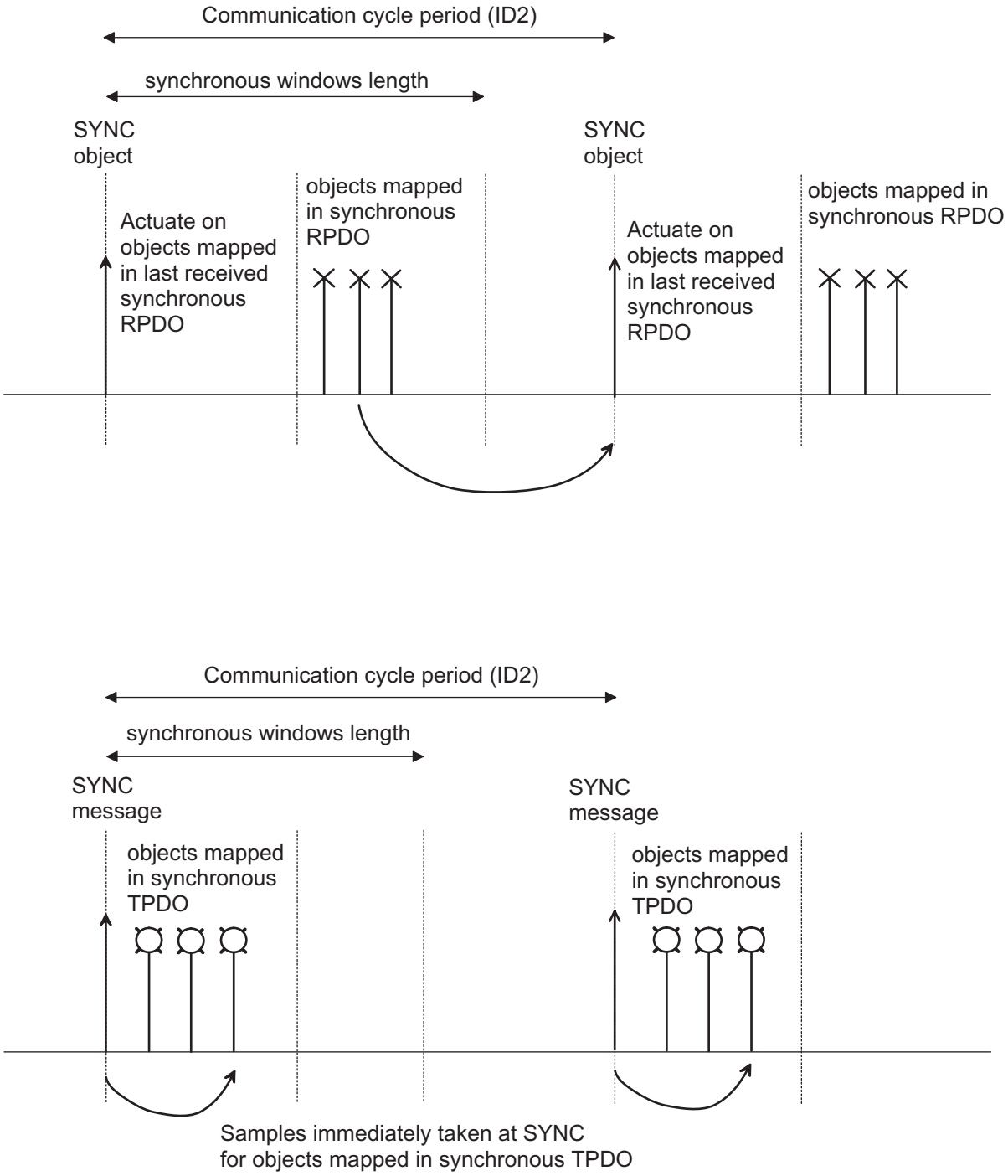


Figure 3-4: BUS SYNChronization and actuation

Additional to the standard can cable (2 signals CAN_H and CAN_L) AMK supports a hardware synchronous signal (CAN_S_H and CAN_S_L) the synchronize the KU clock frequencies to each other. Based on this all synchronous PDOs become active at the same time in every drive. This is important e.g. for transmission of setpoint values, commands, actual values,... .

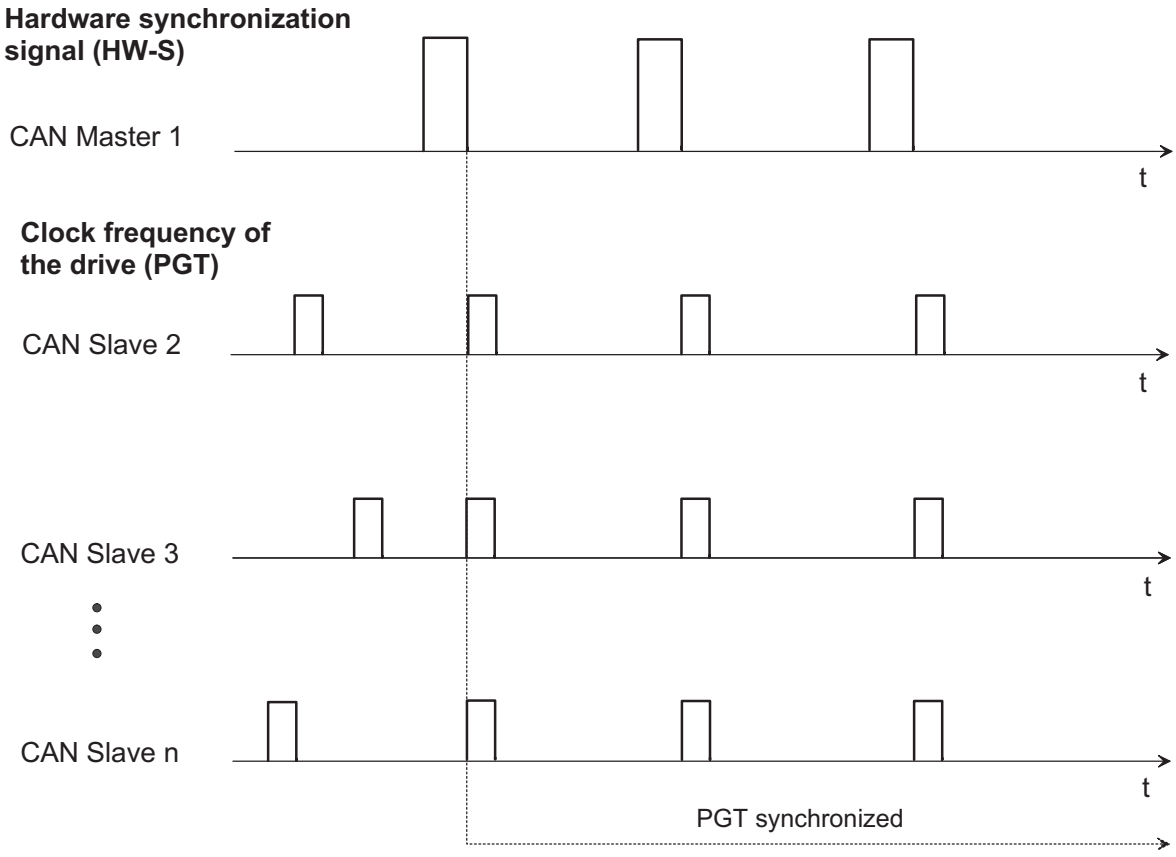


Figure 3-5: Hardware synchronization

4 Important for CAN Network configuration

The configuration of a CAN network consists now in assigning suitable values for the application to the entries of the object list. An important part of this configuration is the assignment of parameters for T-PDOs and R-PDOs. This will be done in the so-called CAN Configuration File.

The following questions must be clarified for the configuration of a CAN network:

- **Which nodes should be subscribers in the network?**
- **Which data should be exchanged between the nodes?**
- **When and how frequently must which data be exchanged?**
- **Which priorities do the data to be exchanged have?**

The answers to these questions result in the values for the parameters of PDOs.

Parameters for T-PDOs and R-PDOs are assigned by means of a CAN configuration file (*.ccf). The contents of the CAN configuration file is a list of value assignments for entries in the object lists of the nodes. This file can be created with a standard Windows text editor.

The "CAN Configuration File" is transmitted through the serial interface to the master AE-PSC card after completion with the AMK auxiliary tools CANconv and DVLader. With these data the CAN master firstly initializes its own dictionary and then the dictionaries of the other network nodes through SDOs (Service Data Object).
The following overview illustrates the procedure.

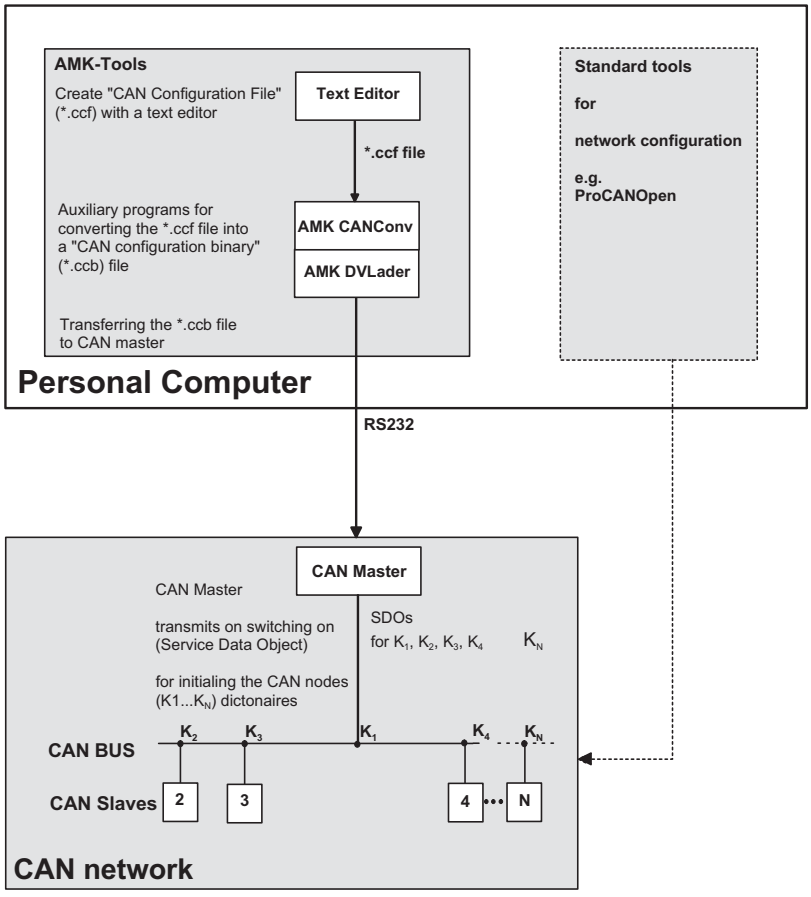


Figure 4-1 Overview of CAN network configuration

5 Predefinition Files

We support 3 predefined files **predefined.ccf**, **confCommon.ccf** and **predefAPI.ccf** which frequently contain the usable definitions and processes.

predefined.ccf contains:
- Definitions of communication indices
- Definitions of COB-IDs from the predefined connection set
- Definitions for use of PS terms for mapping entries
- Definitions of transmission types
- Definitions of frequently used indices
- Definitions of AMK specific indices
- Definitions of complex configuration commands

This file is read at the start of a CAN configuration file with **readFile**.

confCommon.ccf contains:
- Master: automatic setting of index 0x1004 "Number PDOs"
- Master: automatic generation of the necessary client SDO
- Master: setting of index 0x2102 "AMK specific data"
- All slaves: setting of index 0x100c "GuardTime" and index 0x100D "LifeTimeFaktor"

Different symbols must be assigned values for executing **confCommon.ccf.** This file is read at the and of a CAN configuration file with **readFile**.

Additional we support a File which is called **predefAPI.ccf**. This File opens a very comprehensive access to our drive functionality via API parameters which can directly used as mapping entries. API is the KU drive **Ap**plication **i**nterface where the predefAPI.ccf opens access to. PredefAPI.ccf will be read via command readFile **predefAPI.ccf**.

The command **Alias** is used to describe parameters with symbols. Use of these keywords can highly simplify the configuration with many nodes. Special care is necessary in use to avoid unwanted overwriting.

It is naturally possible to create own definitions and processes and to file them in own *.ccf files.

6 Write a Can Configuration File

The following descibes how to create the CAN Configuration File and which entries have to be done to define the transmit and receive PDO´s of your application. The file can be written with every Windows text editor program. See also to the example at the end of this chapter.

6.1 Common parameters

Command	Meaning
nodelist	contains all node numbers
nodegroub slaves	contains all slave nodes
alias BaudrateVal	defines the baudrate [1)]
alias NodeGuardVal	node guarding master checks the presence of all configured slaves 0: node guarding OFF 1: node guarding ON
alias ActivNodesVal	delay time after that the master will initialize the slaves. See ID34026 0: OFF (no delay time) e.g. 0x3000=3sec delay
alias GuardTimeVal	cycle time in which the NMT Master sends request to one slave. Within two requests the slave has to send a confirmation to the master
alias LifeTimeFaktorVal	life guarding Life Time Factor=LTFmaster*number of slave*2 0: OFF

Node guarding
master checks the presence of all configured slaves. Within guardTime one slave is checked.

Life guarding
Life Time=LifeTimeFactor*GuardTime
Slave expected 1 node guarding request from master within the lifetime to check if master is alive

1) Alias BaudrateVal

1000kb
800kb
500kb
250kb
125kb
50kb
20kb
10kb

6.2 Transmit PDO

Command	Meaning
alias master/slave	PDO to master/slave (node n adressing)
alias PDOno	PDO number
alias Map	transmission data
alias TransTyp	transmission type of PDO
alias COBpdo	Identifier of the PDO for adressing and priority of the message (arbitration prinziple)
aliasend	end of transmit PDO
confMasterTransmitPDO	

Example:

```
// Master Configuration
// Transmit PDO

alias master        1           //master is node number 1
alias PDOno         1           //1st PDO of the Can Configuration File
alias Map           OAD0 OAD4   //2 DWORDS are send in CAN synchronous area
alias TransType     CycSync1    //PDO is send after sync signal every ID2 time
alias COBpdo        0x00000201  //COB-ID
confMasterTransmitPDO
```

6.3 Receive PDO

Command	Meaning
alias master/slave	PDO from master/slave (node n adressing)
alias PDOno	PDO number
alias Map	transmission data
alias TransTyp	transmission type of PDO
alias COBpdo	Identifier of the PDO for adressing and priority of the message (arbitration prinziple)
aliasend	end of receive PDO
confMaster/SlaveReceivePDO	

Example:

```
// Slave Configuration
// Receive PDO

alias slave         2           //slave is node number 2
alias PDOno         2           //2nd PDO of the Can Configuration File
alias Map           OED0 OED4   //2 DWORDS are received
alias TransType     CycSync1    //PDO is received after sync signal every ID2 time
alias COBpdo        0x00000201  //COB-ID
confSlaveReceivePDO
```

6.3.1 Mapping entry (alias Map)

To be able to exchange data with the PLC via PDO mapping, the following names are agreed:

Ex	Input byte
EWx	Input word
EDx	Input double word
OEx	Input byte optional module for fast function
OEWx	Input word optional module for fast function
OEDx	Input double word optional module for fast function
Ax	Output byte
AWx	Output word
ADx	Output double word
OAx	Output byte optional module for fast function
OAWx	Output word optional module for fast function
OADx	Output double word optional module for fast function

The relation between PS designators and CAN index/subindex is defined in predefined.ccf which is read in the CAN Configuration File.

6.3.2 Relation of PS designator to the CAN index/subindex

Asynchronous (AFP and I/O) area

Data	Transmit asynchron	Receive asynchron
DWORD	**AD0, AD4, AD8...AD252**	**ED0, ED4, ED8...ED252**
CAN Index Dword	200C sub 1-64	2000 sub1-64
Word	**AW0, AW2, AW4...AW254**	**EW0, EW2, EW4...EW254**
CAN Index Word	200D sub 1-128	2001 sub 1-128
Byte	**A0, A1, A2, A3...A255**	**E0, E1, E2, E3...E255**
CAN Index Byte	200E sub 1-255	2002 sub 1-255

Synchronous area (Fast Function)

Data	Transmit synchron	Receive synchron
DWORD	**OAD0, OAD4, OAD8...OAD60**	**OED0, OED4, OED8...OED60**
CAN Index Dword	200F sub 1-64	2003 sub1-64
Word	**OAW0, OAW2, OAW4...OAW62**	**OEW0, OEW2, OEW4...OEW62**
CAN Index Word	2010 sub 1-128	2004 sub 1-128
Byte	**OA0, OA1, OA2, OA3...OA63**	**OE0, OE1, OE2, OE3...OE63**
CAN Index Byte	2011 sub 1-255	2005 sub 1-255

Binary I/O area

The objects used here are agreed in CiA-DS 401 Device Profile for I/O modules.

CAN object		I/O access
62000108	Index 6000 Subindex 01 size 08 (8-bit)	(output_1_byte 0) 8 bit digital output
60000108	Index 6000 Subindex 01 size 08 (8-bit)	(input_1_byte_0) 8 bit digital input

See also API index table

6.3.2.1 Access to Application Interface (API) inside KU

With the mapping entry it is posible to get access directly to the KU drive interface called API (Application Interface). The mapping entries are defined in the file predefAPI.ccf which is read from the Can configuration file. Every parameter from API can be added to the file predefAPI.ccf according to the API parameter list see chapter API.

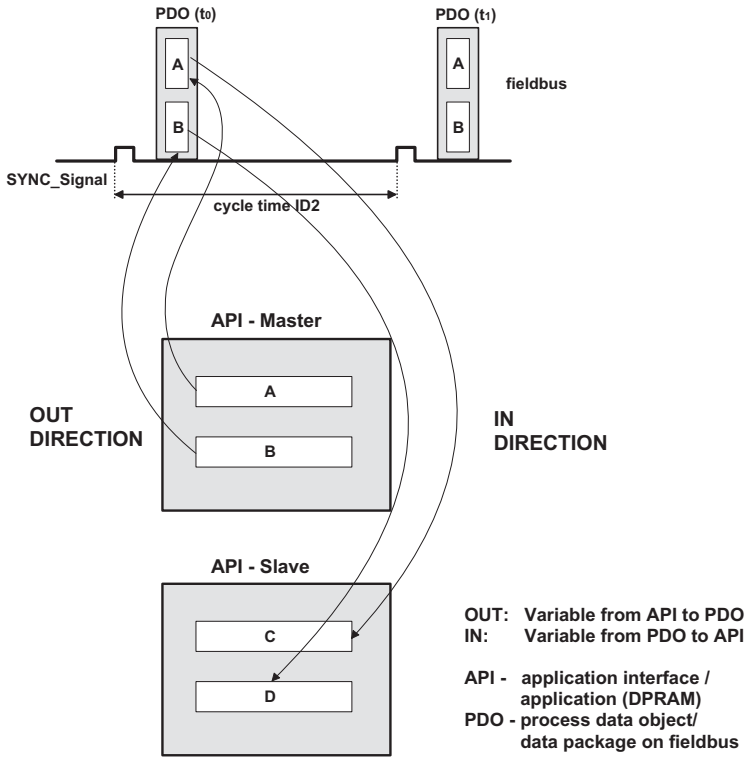


Figure 6-1: Scheme of data transfer

Data consistence, data types

- access to byte and word data (DPRAM READY/BUSY logic)
- access to double word types must be organized in an exact time slot to synchronized signal
- access to semaphores like AFP data is controlled via handshake mechanisms
- Link between CAN (BUS) <-> API is organized always via CAN object dictionary. In fact CAN mapping always works with object dictionary.
  Copy API <-> CAN is performed each interrupt after PGT.

Copy direction

CAN -> API WR
CAN <- API RD

6.4 Index table API

e.g. to receive the actual position of a axis via Can Bus the mapping entry for the receive PDO will be:

alias MAP actual_position_value

length [byte]	CAN name	CAN Index	CAN Sub index	CAN copy direction CAN<->API	Copy time [ms]	
	Setpoints					
4	main_set_point	0x2030	0x01	CAN to API	0.5	1)
4	second_set_point	0x2030	0x02	CAN to API	0.5	1)
2	additional_set_point	0x2030	0x03	CAN to API	0.5	1)
4	main_set_point_RD	0x2031	0x01	API to CAN	0.5	1)
4	second_set_point_RD	0x2031	0x02	API to CAN	0.5	1)
2	additional_set_point_RD	0x2031	0x03	API to CAN	0.5	1)
	Actual values					
2	actual_16_bit_message	0x2040	0x01	API to CAN	0.5	2)
4	actual_32_bit_message	0x2040	0x02	API to CAN	0.5	3)
4	actual_position_value	0x2040	0x03	API to CAN	0.5	
	Bit messages	NU	NU			
2	device_status_bits	0x2048	0x00	API to CAN	5.0	
2	device_control_bits	0x2049	0x00	CAN to API	5.0	
2	real_time_bits	0x204A	0x00	API to CAN	5.0	
1	main_set_point_synchronisation	0x204B	0x00	API to CAN	0.5	
	I/O Area					
1	input_1_byte_0	0x6000	0x01	API to CAN	1.0	4)
1	input_1_byte_1	0x6000	0x02	API to CAN	1.0	4)
1	input_1_byte_2	0x6000	0x03	API to CAN	1.0	4)
1	input_1_byte_3	0x6000	0x04	API to CAN	1.0	4)
1	input_1_byte_4	0x6000	0x05	API to CAN	1.0	4)
1	input_1_byte_5	0x6000	0x06	API to CAN	1.0	4)
1	input_1_byte_6	0x6000	0x07	API to CAN	1.0	4)
1	input_1_byte_7	0x6000	0x08	API to CAN	1.0	4)

length [byte]	CAN name	CAN Index	CAN Sub index	CAN copy direction CAN<->API	Copy time [ms]	
1	input_1_byte_0_WR	0x4000	0x01	CAN to API	1.0	4)
1	input_1_byte_1_WR	0x4000	0x02	CAN to API	1.0	4)
1	input_1_byte_2_WR	0x4000	0x03	CAN to API	1.0	4)
1	input_1_byte_3_WR	0x4000	0x04	CAN to API	1.0	4)
1	input_1_byte_4_WR	0x4000	0x05	CAN to API	1.0	4)
1	input_1_byte_5_WR	0x4000	0x06	CAN to API	1.0	4)
1	input_1_byte_6_WR	0x4000	0x07	CAN to API	1.0	4)
1	input_1_byte_7_WR	0x4000	0x08	CAN to API	1.0	4)
1	output_1_byte_0	0x6200	0x01	CAN to API	1.0	5)
1	output_1_byte_1	0x6200	0x02	CAN to API	1.0	5)
1	output_1_byte_2	0x6200	0x03	CAN to API	1.0	5)
1	output_1_byte_3	0x6200	0x04	CAN to API	1.0	5)
1	output_1_byte_4	0x6200	0x05	CAN to API	1.0	5)
1	output_1_byte_5	0x6200	0x06	CAN to API	1.0	5)
1	output_1_byte_6	0x6200	0x07	CAN to API	1.0	5)
1	output_1_byte_7	0x6200	0x08	CAN to API	1.0	5)
1	output_1_byte_0_RD	0x4200	0x01	API to CAN	1.0	5)
1	output_1_byte_1_RD	0x4200	0x02	API to CAN	1.0	5)
1	output_1_byte_2_RD	0x4200	0x03	API to CAN	1.0	5)
1	output_1_byte_3_RD	0x4200	0x04	API to CAN	1.0	5)
1	output_1_byte_4_RD	0x4200	0x05	API to CAN	1.0	5)
1	output_1_byte_5_RD	0x4200	0x06	API to CAN	1.0	5)
1	output_1_byte_6_RD	0x4200	0x07	API to CAN	1.0	5)
1	output_1_byte_7_RD	0x4200	0x08	API to CAN	1.0	5)
	Error Bits					
2	error_bits	0x204C 0x204C	0x01 0x02	API to CAN CAN to API		6)
	Analog Area					
2	analog_out_1	0x6411	0x01	CAN to API	1.0	9)
2	analog_out_2	0x6411	0x02	CAN to API	1.0	9)
2	analog_out_3	0x6411	0x03	CAN to API	1.0	9)
2	analog_out_4	0x6411	0x04	CAN to API	1.0	9)
2	analog_out_1_RD	0x4411	0x01	API to CAN	1.0	9)
2	analog_out_2_RD	0x4411	0x02	API to CAN	1.0	9)
2	analog_out_3_RD	0x4411	0x03	API to CAN	1.0	9)
2	analog_out_4_RD	0x4411	0x04	API to CAN	1.0	9)
1	actual_position_referenced	0x204D	0x00	API to CAN	5.0	
	Setpoint Source	NU	NU			
4	setpoint_source1	0x2050	0x01	API to CAN		7)
4	setpoint_source2	0x2050	0x02	API to CAN		7)
4	setpoint_source3	0x2050	0x03	API to CAN		7)
4	setpoint_source4	0x2050	0x04	API to CAN		7)
4	setpoint_source1_WR	0x2051	0x01	CAN to API		7)
4	setpoint_source2_WR	0x2051	0x02	CAN to API		7)
4	setpoint_source3_WR	0x2051	0x03	CAN to API		7)

length [byte]	CAN name	CAN Index	CAN Sub index	CAN copy direction CAN<->API	Copy time [ms]	
4	setpoint_source4_WR	0x2051	0x04	CAN to API		7)
	AFP area					
8	AFP write block				5.0	10)
8	AFP read block				5.0	11)

- 1) function of operating mode (ID32800...)
- 2) function of AW16 message (ID32785)
- 3) function of AW32 message (ID32786)
- 4) every input byte shows the state of 8 internal binary inputs, actual copy direction API to CAN (up to version 0501) and depending on mapping for next software versions.
- 5) every output byte writes to 8 internal binary outputs, actual copy direction API to CAN (up to version 0501) and depending on mapping for next software versions.
- 6) error_bits are copied in asynchronous time not cyclic
- 7) copy time is depending on ID2 „SERCOS cycle time“
- 9) actual copy direction API to CAN (up to version 0501) and depending on mapping for next software versions.
- 10) AFP write block only accessible through dict. object 0x2021 sub1 to sub8.
- 11) AFP read block only accessible through dict. object 0x2020 sub1 to sub8.

All this functionality is valid with PSC software **PSC 1.01 4800** and KU version **KU 1.11 3900**.

6.4.1 Transmission Type (alias TransTyp)

TransType	Meaning
CycSync x	Transmission rate of x means that the **synchronous PDO** message is transmitted with every x-th SYNC object. The cycle time of the SYNC object is ID2 SERCOS cycle time. e.g. ID2=2ms, CycSync3 : transmitted every 6ms (commands, setpoints, actual values,...)
AMKevent	**Asynchronus PDO**; PDO will be transmitted only if data changed without any relation to the SYNC object. (AFP or I/O transmission)

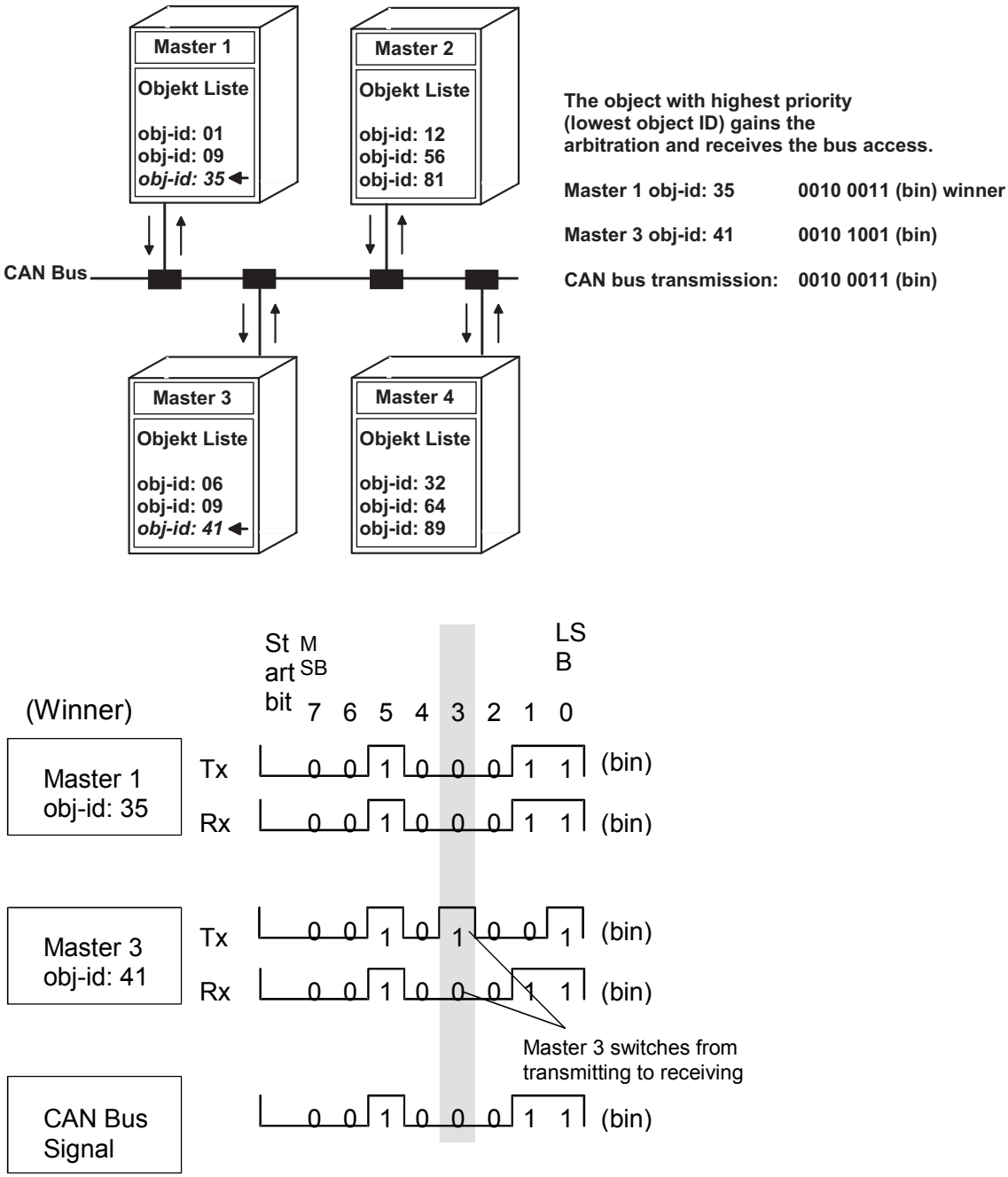
6.4.2 COB-ID and arbitration prinziple (alias COBpdo)

The CAN bus becomes a network in that messages of different priority are transmitted by broadcasting between all subscribers. The key concept is the so-called arbitration system which regulates the bus accesses of each subscriber.

If a subscriber wants to send a message (PDO), then it sends a telegram header when the bus is free. The header contains an 11-bit communication object identifier (obj-id,COB-ID) which is assigned specifically to this message. The lowest significant object identifier has the

highest priority, gains the arbitration and receives the bus access. Its message is sent without loss of time.

Figure 6-2 Arbitration principle



7 CCF File according to a example application

The CAN network has a master KU (node 1) a slave KU (node 2) and an external I/O module with 32 inputs and outputs (node 5).

The following figure defines which data will be exchanged between the nodes.

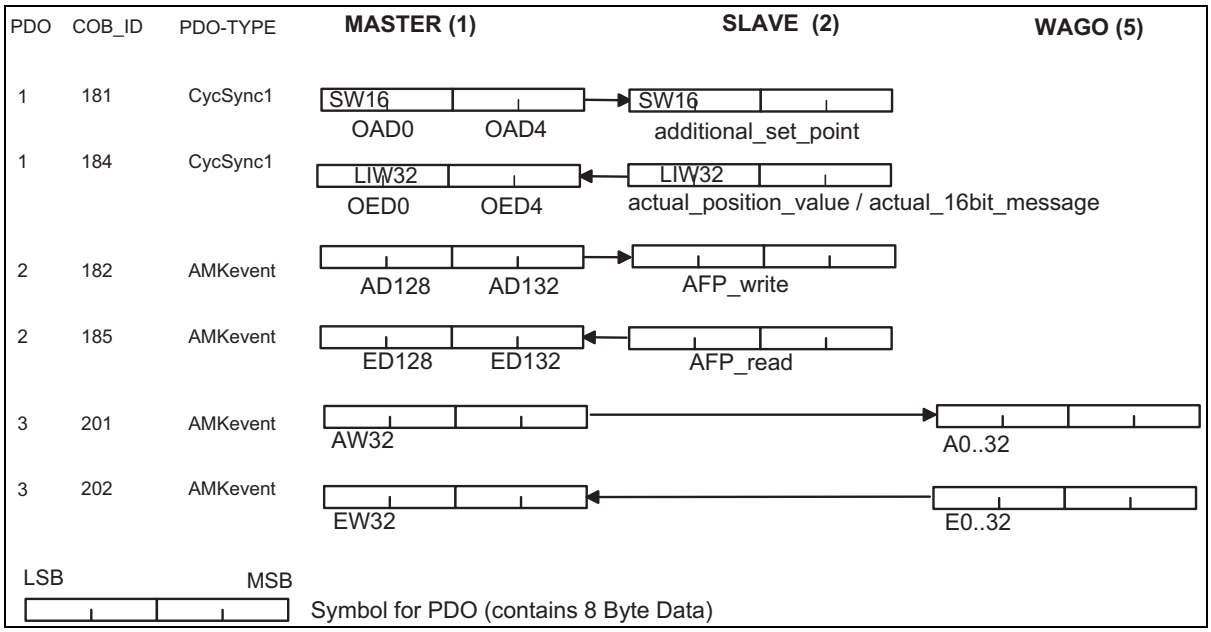


Figure 7-1: Data exchange definition

The following CCF-File describes the application according to the above definitions. This file needs to be downloaded to the master with the tools CANConv and DVLader.

```
// ================================================================
//   Filename: example01.ccf//
//   Date: 12.09.01
//================================================================
// History
// 100901
// ================================================================
      readFile predefined.ccf
      readFile predefAPI.ccf

//------------------------
//   Common parameters
//------------------------
      nodelist 1  2  5
      nodegroup slaves   2  5

      alias BaudrateVal            1000kb
      alias NodeGuardVal          0      //0 off    1 on
      alias ActivNodesVal         0      //0 off 0x10 on
      alias GuardTimeVal          0
      alias LifeTimeFaktorVal     0
```

```
//------------------------
//   Master Configuration
//------------------------

     alias master        1

//   1.Transmit PDO                        to Slave No2 (KU)
      alias PDOno       1
      alias Map         OAD0 OAD4
      alias TransTyp    CycSync1
      alias COBpdo      0x00000181
      aliasend

      confMasterTransmitPDO

//   1.Transmit PDO                TPDO to Slave No2 (KU)
      alias PDOno       2
      alias Map         AD128 AD132      //AFP write block
      alias TransTyp    AMKevent
      alias COBpdo      0x00000182
      aliasend

      confMasterTransmitPDO

//   1.Transmit PDO                TPDO to Slave No5 (WAGO I/O)
      alias PDOno       3
      alias Map         AW32
      alias TransTyp    AMKevent
      alias COBpdo      0x00000201
      aliasend

      confMasterTransmitPDO

//   1. Receive PDO  from Slave No2 (KU)
      alias PDOno       1
      alias Map         OED0 OED4        //actual_position, actual_16bit_message
      alias TransTyp    CycSync1
      alias COBpdo      0x00000184
      aliasend

      confMasterReceivePDO

//   1. Receive PDO  from Slave 1
      alias PDOno       2
      alias Map         ED128 ED132      //AFP read block
      alias TransTyp    AMKevent
      alias COBpdo      0x00000185
      aliasend

      confMasterReceivePDO

//   1. Receive PDO  from Slave No5 (Wago I/O)
      alias PDOno       3
      alias Map         EW32
      alias TransTyp    AMKevent
      alias COBpdo      0x00000202
      aliasend

      confMasterReceivePDO
```

```
//------------------------
//   Slave 2 Configuration
//------------------------

      alias slave        2

//   1.Transmit PDO
      alias PDOno        1
      alias Map          actual_position_value  actual_16bit_message
      alias TransTyp     CycSync1
      alias COBpdo       0x00000184
      aliasend

      confSlaveTransmitPDO

//   1.Transmit PDO
      alias PDOno        2
      alias Map          AFP_read
      alias TransTyp     AMKevent
      alias COBpdo       0x00000185
      aliasend

      confSlaveTransmitPDO

//   1. Receive PDO  from Master
      alias PDOno        1
      alias Map          additional_set_point
      alias TransTyp     CycSync1
      alias COBpdo       0x00000181
      aliasend

      confSlaveReceivePDO

//   1. Receive PDO  from Master
      alias PDOno        2
      alias Map          AFP_write
      alias TransTyp     AMKevent
      alias COBpdo       0x00000182
      aliasend

      confSlaveReceivePDO

//---------------------------------------
//   Slave 5 Configuration Wago EA Modul
//---------------------------------------

      alias slave        5

//  1. Transmit PDO to Master
      alias PDOno        1
      alias Map          input_1_byte_0    input_1_byte_1    input_1_byte_2    input_1_byte_3
      alias TransType    AMKevent
      alias COBpdo       0x00000202
      aliasend

      confSlaveTransmitPDOnew

//  2. Receive PDO  from Master
      alias PDOno        2
      alias Map          output_1_byte_0    output_1_byte_1    output_1_byte_2    output_1_byte_3
```

```
    alias TransTyp    1
    alias COBpdo      0x00000201
    aliasend

    confSlaveReceivePDOnew

    readFile confCommon.ccf

/*================================= EOF============================*/
```

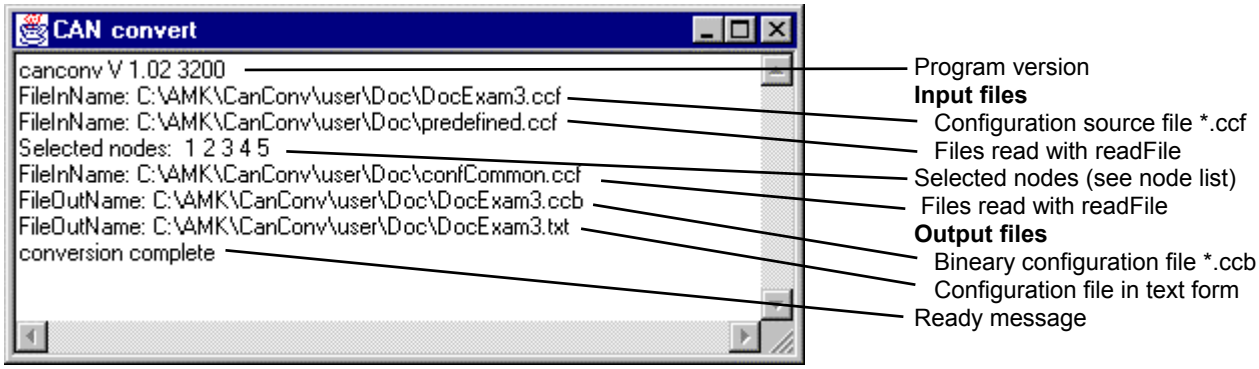
8 Converting with AMK tool CANConv

The following operations are performed in the conversion of a configuration source file (*.ccf) into a configuration binary file (*.ccb):

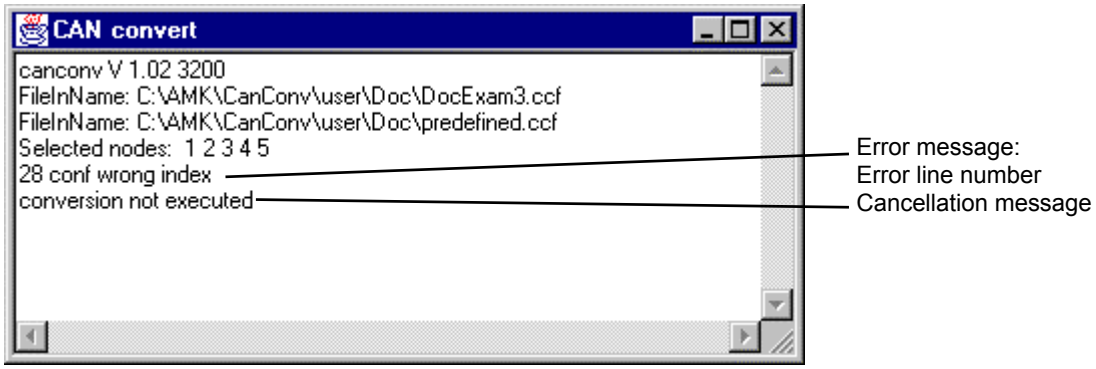
- Creation of the entries according to regulations in the configuration source file
- However no configuration data are contained for nodes which are listed in the **nodelist**, stands in the configuration binary file for number of supported entries: 0
- Configuration data for nodes which stand in **nodelist** are removed
- Sorting the entries according to node number
- Sorting the entries in node 1 = Master according to index and subindex
- Testing for double entries for index/subindex of a node
  Note: Double entries generated by **PDO** for **[COBID]** and the number of the mapping entries are ignored
- Output of the result in the configuration binary file (*.ccb)
- Transformation of the configuration binary file (*.ccb) into a text file *.txt
- If errors are determined, then the conversion is aborted and there is an error output

8.1 Results message

Screen display for errorfree conversion (example)



Screen display on faulty configuration source file (example):



8.2 Installation and selection of CANConv

The installation of all required files including examples is performed by selecting **setupCanConvert.exe**.

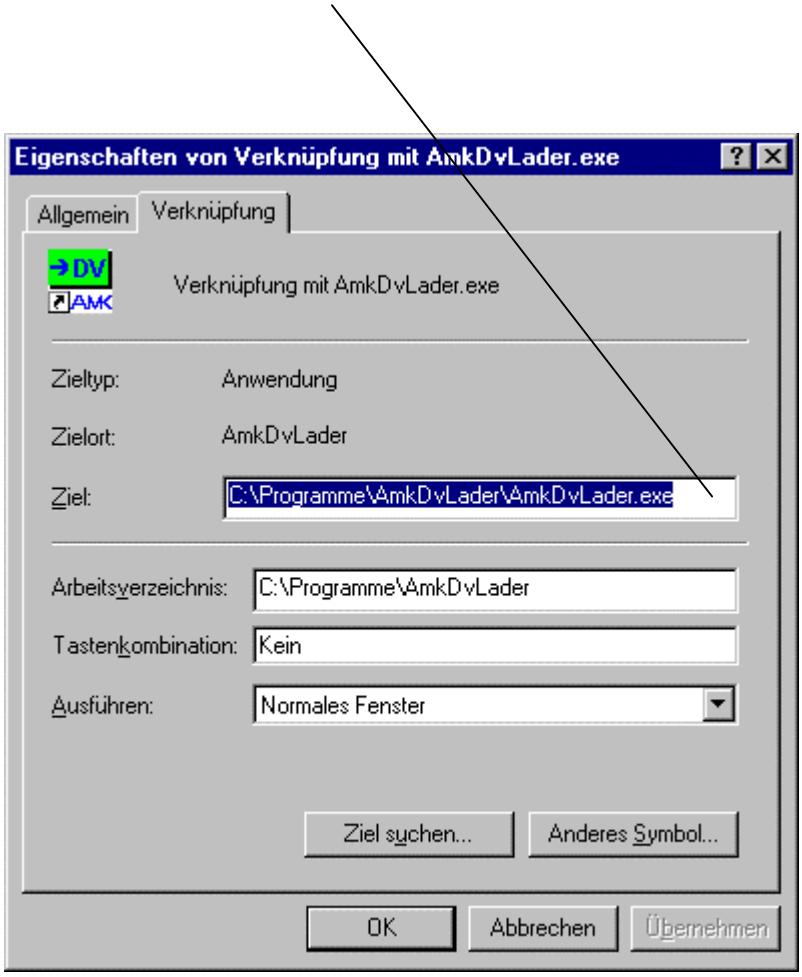
Standalone mode
Selecting **Canconv.bat** starts the application and initially shows a file selection dialog. After a file of the type *.ccf has been selected, the conversion starts automatically. After completed conversion (see results message)
the window can be closed again. On frequent use you are recommended to create a link with **Canconv.bat** on the desktop. To avoid visibility of the DOS window, the properties "Execute as symbol" and "Close on exiting" should be assigned to it (only Win95/Win98).

Linking with AmkDvLader
The configuration binary files *.ccb created in the conversion must be loaded on the CAN master for your application with the aid of the AmkDvLader program. Canconv.bat can also be selected by AmkDvLader. For this purpose the selection of AmkDvLader must be supplemented by the following parameters (see also AmkDvLader description).

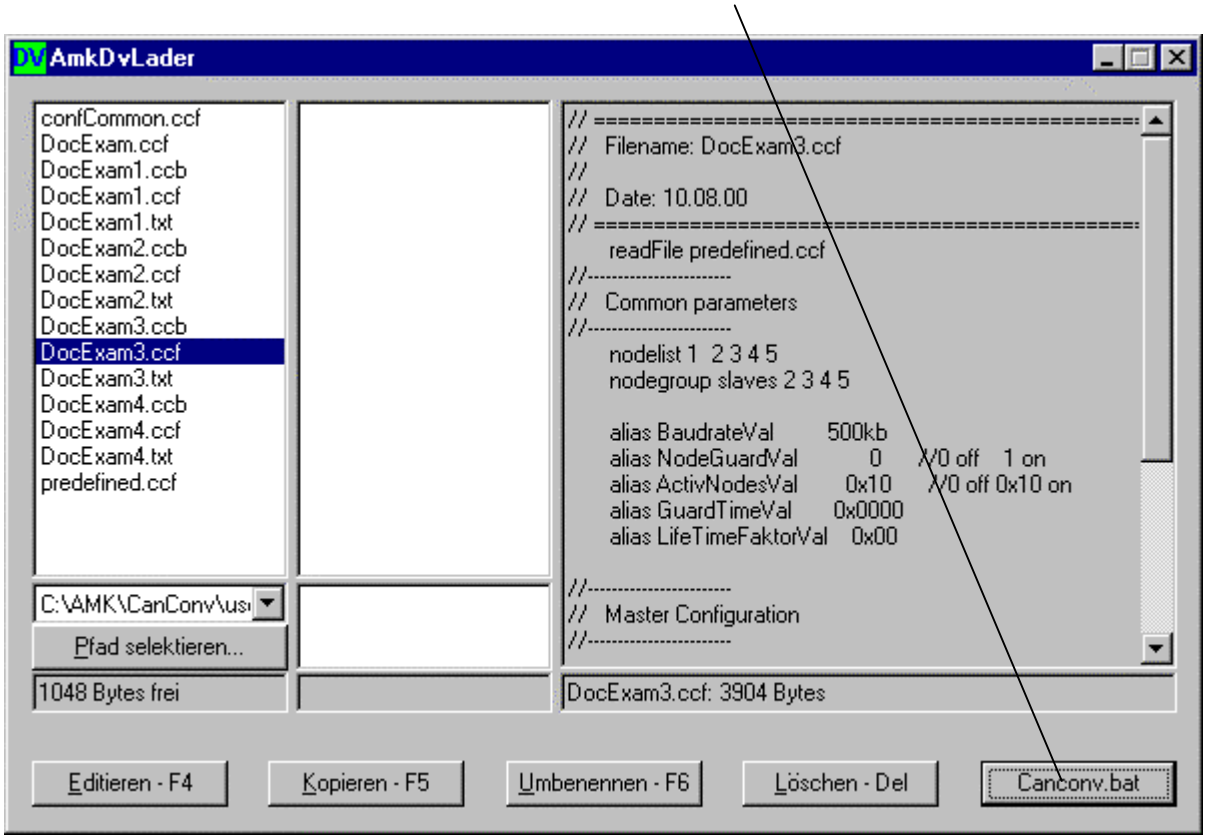
-tccf -cC:*YOUR_PATH*\CanConv\Canconv.bat -eccf

e.g.

-tccf -cC:\Programme\CanConv\Canconv.bat -eccf



AmkDvLader now receives an additional button "Canconv.bat".



The button is activated if a file of the type *.ccf is selected. The conversion of the selected *.ccf file is started by activating the "Canconv.bat" button. After successful conversion the created *.ccb file can be loaded immediately to the CAN master by activating "Copy - F5".

9 AMK tool DVLader

9.1 Introduction

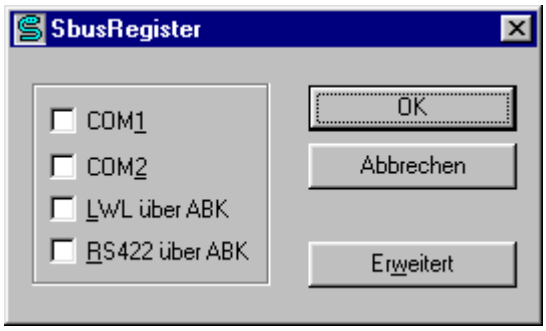
The program described here is a Windows program for access to file management systems (DV) on MC and PS cards which can be reached through Sbus. It has the following properties:

- Simultaneous display of all files of a selected PC path and a selected DV (2-window display)
- Fast view of the contents of a selected file (PC or DV) as binary file or for known file types as text file
- Functions for copying between PC and DV (both directions) as well as for deleting and renaming files in the PC or in the DV
- Display of the current Sbus topology (all active interfaces as well as the DVs connected to them)
- The program works under Window 95®, Windows 98® and Windows NT®. The Sbus can always be used through Com interface; the AB-K02 card does **not** function under **Windows NT**!
- There is a freely assignable special function for selecting an arbitrary program.

The general principles of Windows operation are presupposed in this description.

9.2 Configuring Sbus

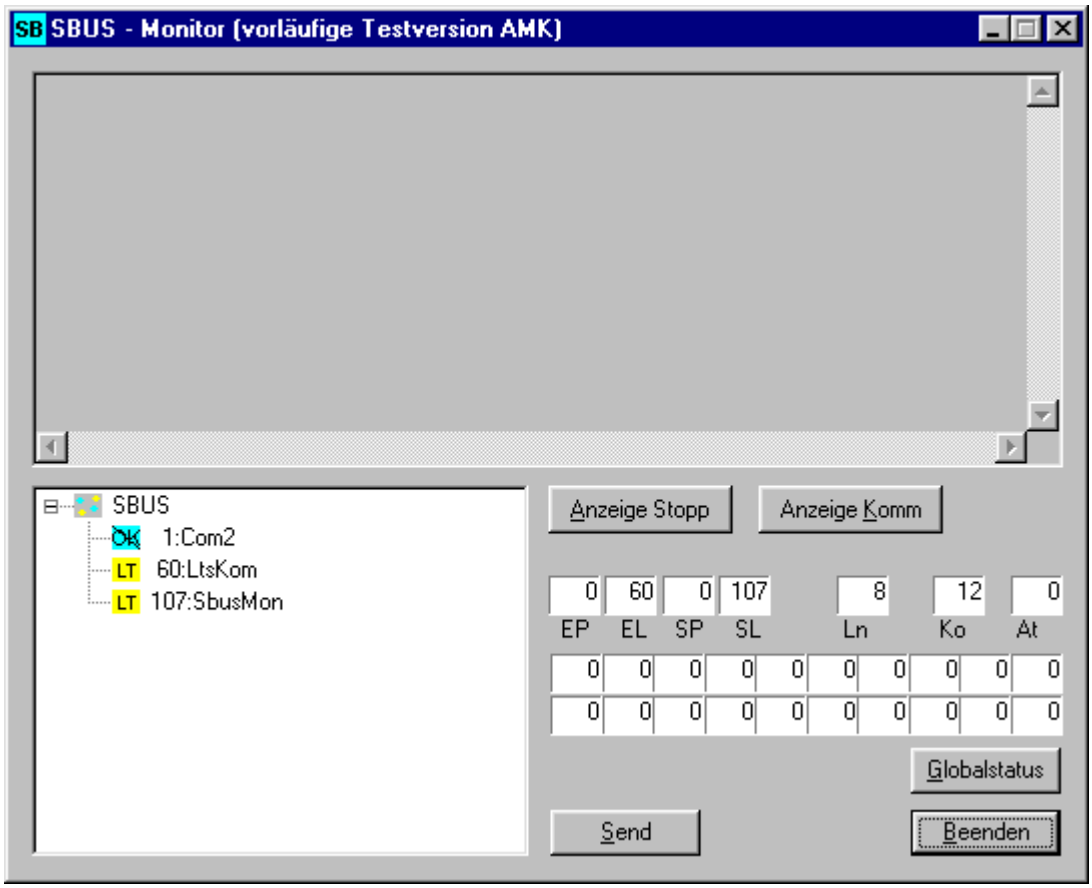
So that it is possible to work with the Sbus, its physical interfaces (physical ports) must be defined and configured. This is done using the "SbusRegister" program which is enclosed with the installation.



In the normal case only the required COM port 1 or 2 must be selected and confirmed by OK button. The two other options concern bus interfaces through additional special hardware. Other Com interfaces can be set using the "Advanced" button if available in the current PC.

The settings which are made here are valid globally for all programs running on this PC. They must be made generally before selecting the DvLader.

The "SbusDialog" program also enclosed is used for testing the Sbus connection.



It is possible to test with this whether a connection between PC and the opposite number can be made with the selected settings. The connection changes the icon of the Com port into the OK state.

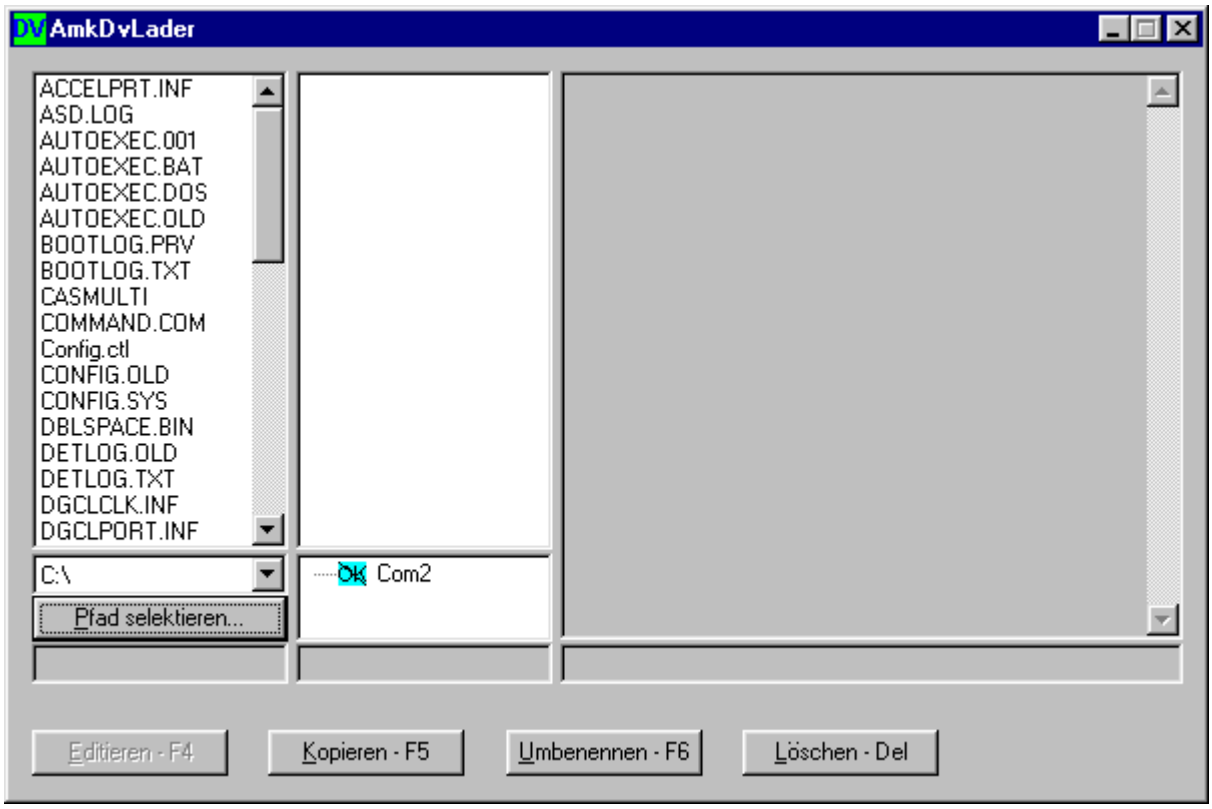
Note!

This test program cannot run jointly with another Sbus application in the normal case. Therefore close the monitor before starting the DvLader!

9.3 Operation

9.3.1 Starting the program

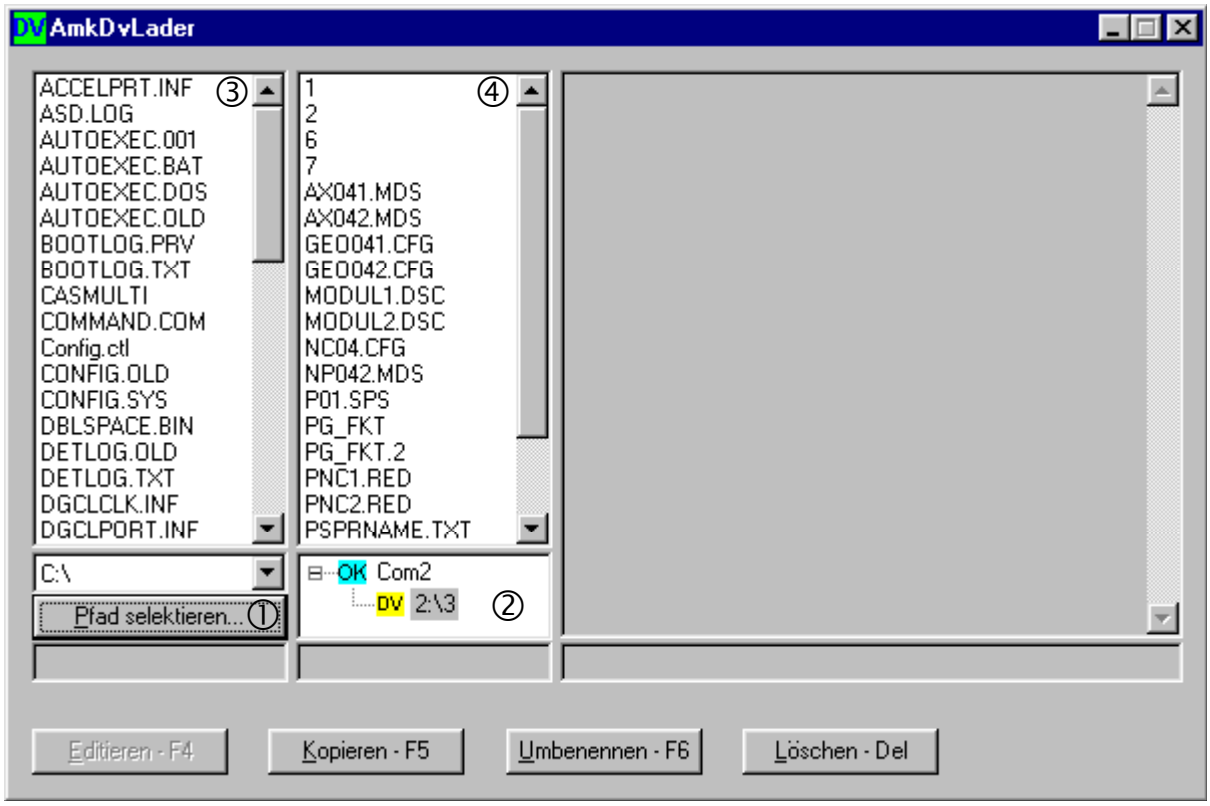
The following main window appears after selecting the program:



The workspace is divided into three vertically divided parts:

- The left column shows the selected PC path and the files contained in it
- The central column is used for displaying in each case a selected DV and the files contained in it
- The right, largest part serves for displaying the contents of a file marked in one of the two left column (fast view)

After a few seconds the Sbus goes into the OK state and the DV and the files contained in it are displayed in the central column:



9.3.2 Selection of the working path in the PC

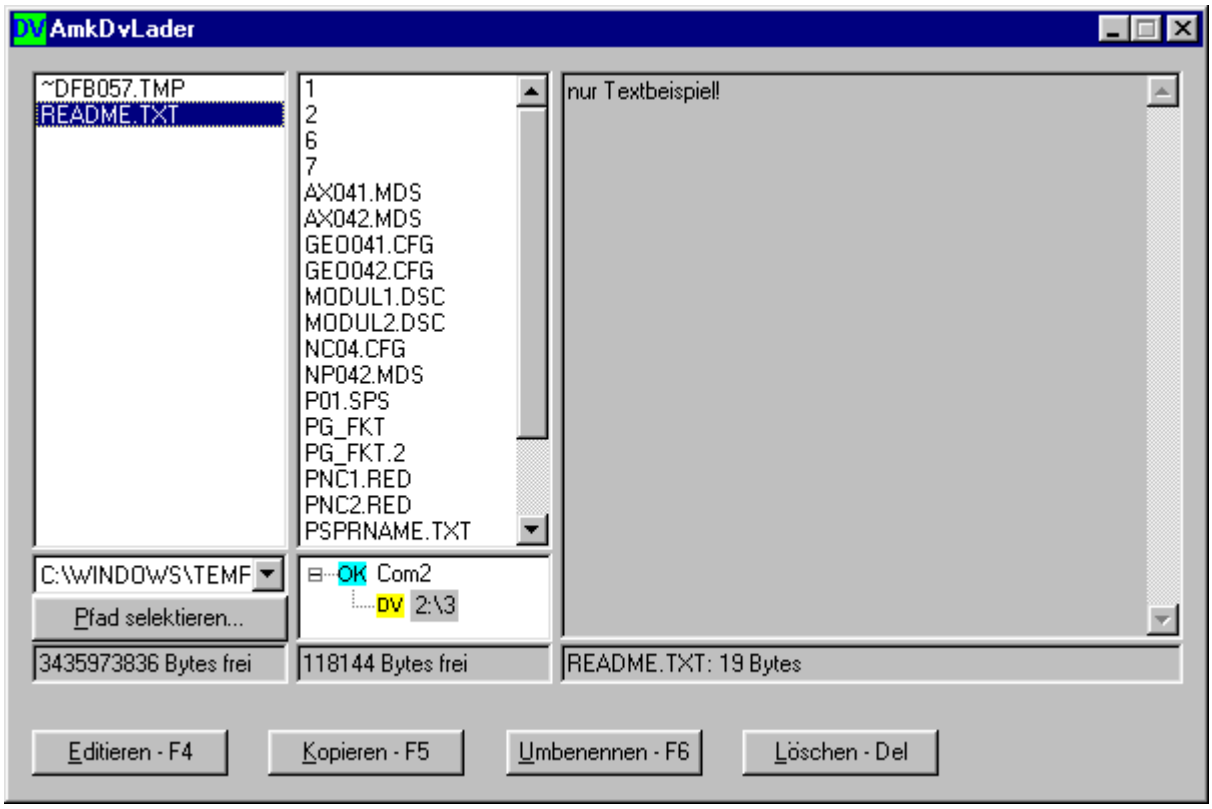
A selection dialog with which the PC working path can be selected appears after activating the "Select path..." button ①.

9.3.3 Selection of the DV

In the normal case only one DV is available on the Sbus. This is then selected automatically. If several DVs are active, the required DV must be selected by the operator by clicking in the tree diagram ②.

9.3.4 Selection of the file to be edited

To be able to work with the actual DV function, a file must be selected in the PC ③ or in the DV ④. Only one of the files is always ready for editing either in the PC or in the DV. The contents of the selected file are displayed in the window of the fast view. The display is made in text form if the file is recognized as text file, or binary for all other file types. Only the first 1000 bytes of the file are displayed in each case and the remainder is cut off.



9.3.5 Editing and copying a file

The buttons at the bottom of the window refer generally to the file selected in the PC or in the DV:

- Edit: The selected file is transferred to the Windows system editor for editing. This function works only with the files available on the PC.
- Copy: The selected file is copied from the PC to the DV or vice versa.
- Rename: The selected file can be remained.
- Delete: The selected file is deleted (after a prompt).

9.4 Configuration possibilities through command line parameters

9.4.1 Extension of the list of the text file formats

The following file types are always interpreted as text files:

- .red
- .ini
- .txt

This list can be extended as desired using the command line parameter "-t". The file types .doc and .bat are displayed in addition as text files by selecting the program with the parameters

DvLader -tdoc -tbat

Caution: There must be no space between the command and the file type!

9.4.2 Definition of a special function

The DvLader program can be extended by the selection of a special function. An executable file which implements the function is determined by the command "-c". The file types which can edit the special function are defined with the command "-e". The special function refers like all other functions always to the selected file. However, it works only with files available on the PC.

In the following case

DvLader -cc:\programme\msoffice\winword\Winword.exe -edoc -etxt

the special function becomes active if the selected file has the type .doc or .txt.