AMK

AMKASYN

VARIABLE SPEED DRIVES

AMKASYN

Digital drive systems

CAN Network configuration

- **Series KE/KW**
- **Series KU**
- **SYMAC**

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.

PRELIMINARY

2002/27.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-PLC	**A**MKASYN **E**xtension **PL C**AN Option card
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
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	**Concise c**onfiguration **b**inary file type *.ccb
ccf	**Concise 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
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-PLC 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.

The synchronization is done by the AMK hardware synchronization signal.

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:

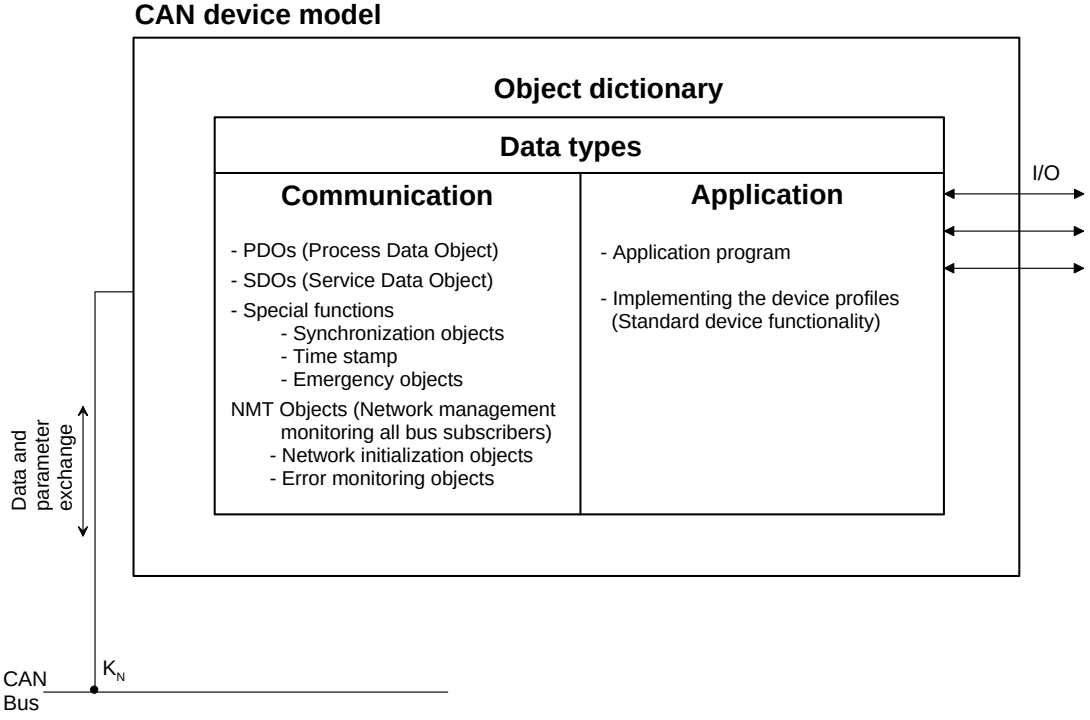
- CANopen Minimum Capability Device Boot-Up
- Node Guarding / Life Guarding
- Transmit PDOs
- Receive PDOs
- Client/Server SDOs
- Emergency Object
- Synchronization Object

3.1 Object Dictionary

Each CANopen device has a CANopen object dictionary. 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 dictionary. Access to entries of the object dictionary is made through a 16-bit index and an 8-bit subindex.

The object dictionary 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.

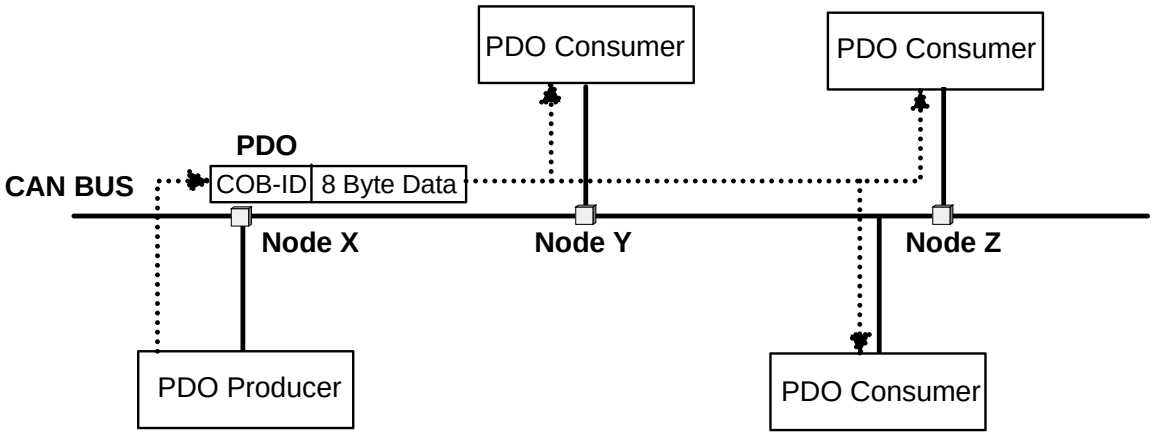
Figure 3-1 CANopen communication model



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 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 dictionary which determines the communication properties of a node (see object dictionary 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, DeviceEvent)

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 period. The synchronisation object carries no data and is easy to generate. After SYNC object synchronous messages will be send. Asynchronous PDOs will be send after Synchronous. Asynchronous TPDOs are transmitted without any relation to SYNC. The data of asynchronous RPDOs can be transferred with delay to the application but is always consistent.

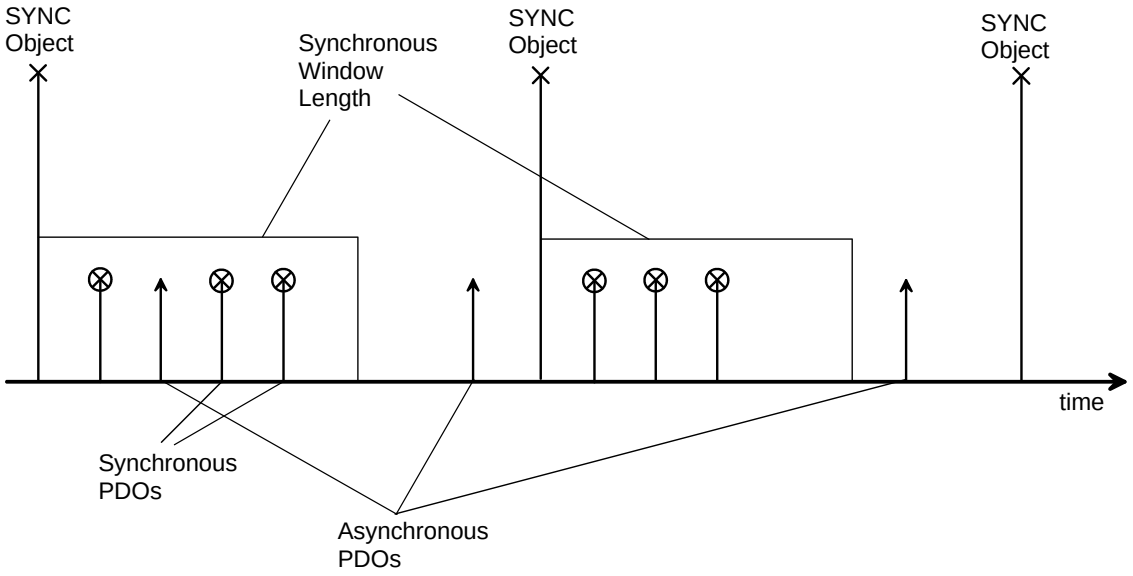


Figure 3-3: Synchronous and Asynchronous PDO transmission

The jitter of this SYNC object 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.

To avoid SYNC jitter AMK CAN devices use additional hardware line named Hardware-SYNC, which synchronizes all synchronous data in all devices linked to the CAN network.

Received synchronous messages will be actuated after the next Hardware SYNC signal. After actuation the synchronous messages become active. The following figure shows the prinziple of SYNC PDOs transmission.

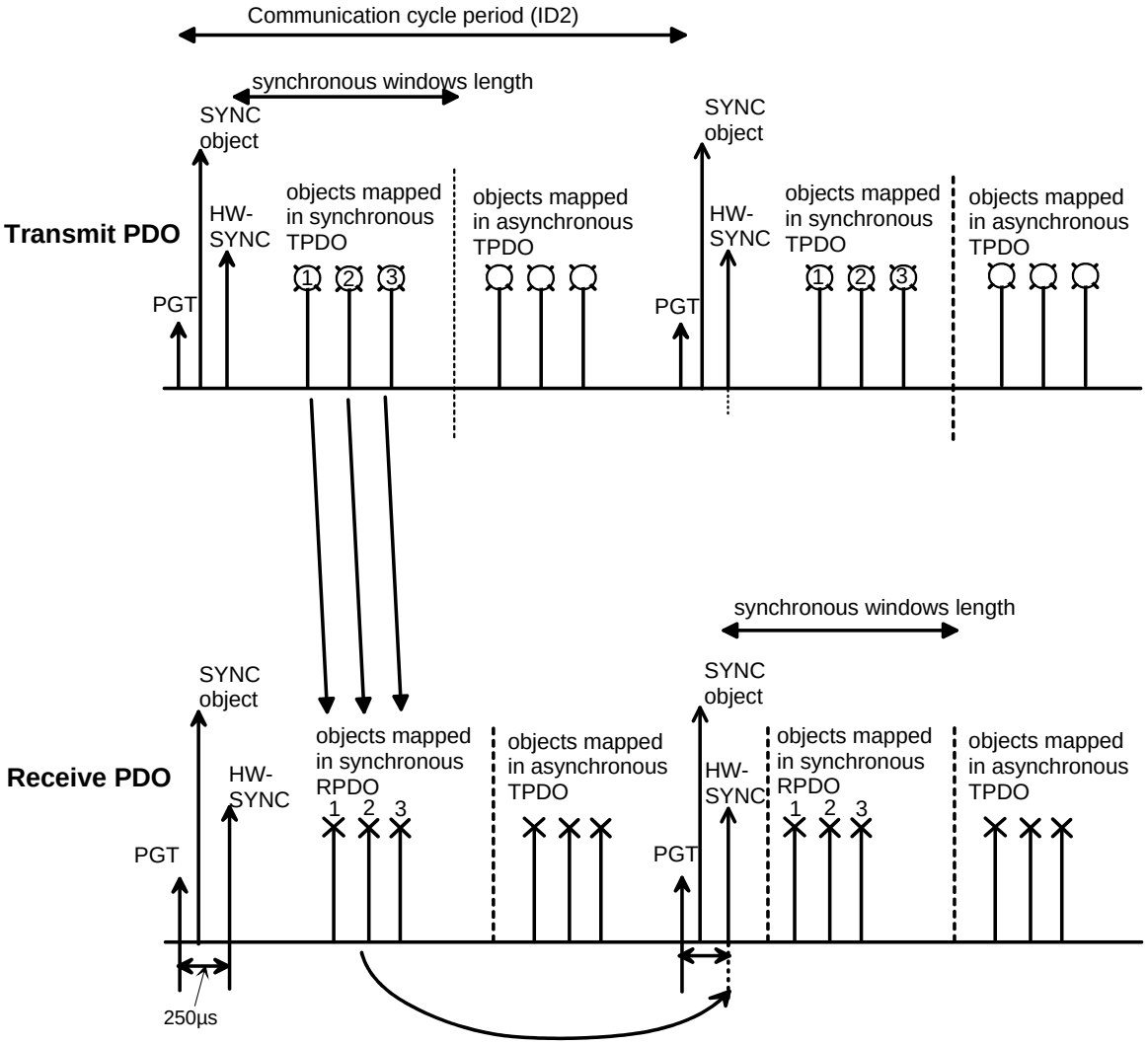


Figure 3-4: BUS SYNChronization and actuation

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) for synchronization of PGT signal in all devices in the network. 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,... .

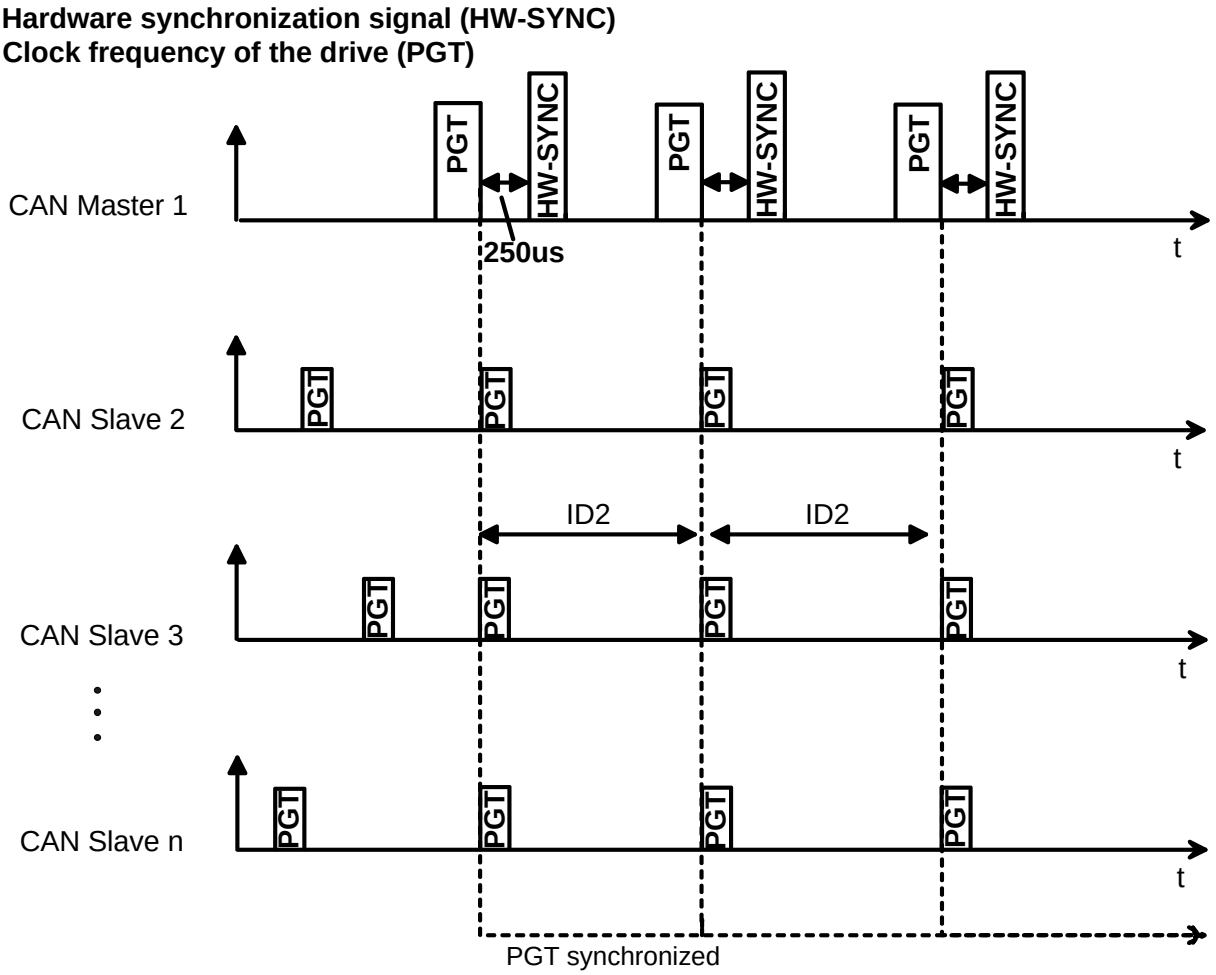


Figure 3-5: Hardware synchronization

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 dictionary. 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 Device Configuration File (DCF) or in concise variants- CONCISE DCF (AMK-CCF-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 the 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 CONCISE CONFIGURATION FILE (*.ccf). The contents of the concise configuration file is a list of value assignments for entries in the object dictionary of the nodes. This file can be created with a standard Windows text editor.

The "Concise Configuration File" must be converted into a concise configuration binary file (CCB-File), which is transmitted through the serial interface to the master AE-PLC card with AMK tool. 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) after next power on.
The following overview illustrates the procedure exemplified by KU System.

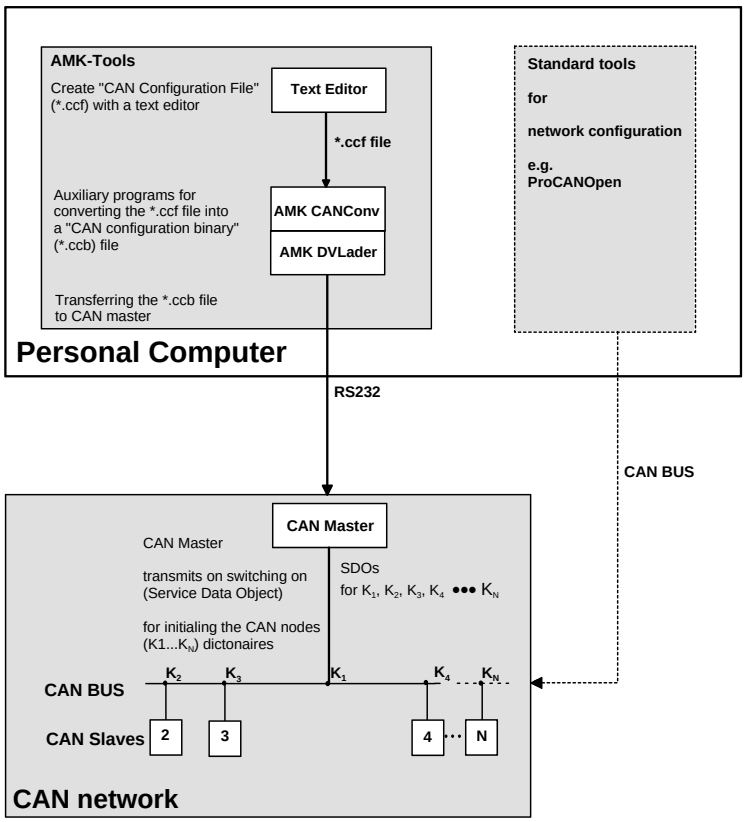


Figure 4-1 Overview of CAN network configuration

5 Predefinition Files

We support 3 predefined files **predef2.ccf**, **confCommon.ccf** and **predefAPI.ccf** which contain the usable definitions and processes.

Predef2.ccf contains:
- Definitions of communication indices
- Definitions of COB-IDs from the predefined connection set
- Definitions for use of PLC 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** command.

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 be directly used as mapping entries. API is the 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 Concise Configuration File (CCF)

The following descibes how to create the Concise 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 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
nodegroup slaves	contains all slave nodes
alias BaudrateVal	defines the baudrate [1)]
alias NodeGuardVal	Must be there for compatibility but actually switched OFF 0: OFF 1: 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	Life Guarding : Slave expects request from master every Life Time
alias LifeTimeFaktorVal	=LTFmaster*number of slave*2 0: OFF

Node guarding (master)
The Guarding Time is the result of GuardTimeVal x LifeTimeFaktorVal. Within this time the master expected an answer of one slave. Within guardTimeVal the master sends a request to one slave. If the answer of a slave is missed after 2xGuardTime the master will become error state.
eg.

```
// Node Guarding for node 1 master
conf 1 0x100C 0 WORD 1000        // master sends request to one slave every 1000ms
conf 1 0x100D 0 BYTE 1           //LTF master=1
```

Life guarding (slave)
Life Time=GuardTimeVal x LifeTimeFaktorVal
Within the Life Time every slave expected 1 node guarding request from master to check if master is alive. If guarding request is missed the slave will become an error state. Life guarding is only active after starting node guarding.

eg.

```
// Life Guarding for node 2 slave
conf  2 0x100C 0 WORD 1000       //slave erwartet Abfrage alle 4000ms
conf  2 0x100D 0 BYTE 4          //eg. 2 slaves in network
```

1) Alias BaudrateVal

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

6.2 Data exchange via PDO

6.2.1 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 InhibitTime	Time to disable this 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 in the master object dictionary
alias Map           QD_S0  QD_S1    //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
```

6.2.2 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 in the slave object dictionary
alias Map            ID_S0  ID_S1   //2 DWORDS are received
alias TransType      CycSync1       //not considered for R-PDO
alias COBpdo         0x00000201     //COB-ID
confSlaveReceivePDO
```

6.3 Mapping Entry (alias Map)

To be able to exchange data with PLC via PDO mapping, the following names and adresses are agreed.
IMPORTANT: Please take care that no address overlapping is allowed!

The relation between PLC designators and CAN index/subindex is defined in the AMK file "predef2.ccf" which is read in the CAN Configuration file.

6.3.1 CAN binary I/O AREA

Output PLC to CAN Transmit PDO		Input PLC from CAN Receive PDO	
QBx:	BYTE	IBx:	BYTE
QWx	WORD	IWx:	WORD
QDx	DWORD	IDx	DWORD

Index Byte	8	9	10	11	12	13	14	15		60	61	62	63
Index Word	4		5		6		7		...	30		31	
Index DW	2				3				..	15			

Bytes 0...7 are reserved for AMKASYN I/O option cards

6.3.2 AFP AREA (asynchronous)

Output PLC to CAN Transmit PDO		Input PLC from CAN Receive PDO	
QLx:	LONG WORD (8byte)	Ilx:	LONG WORD (8byte)

Index LW*	16	...	31

* only index IL/QL 16 – 31 are defined for AFP client

6.3.3 SYNC AREA (synchronous)

Output PLC to CAN Transmit PDO		Input PLC from CAN Receive PDO	
QB_Sx:	BYTE	IB_Sx:	BYTE
QW_Sx:	WORD	IW_Sx:	WORD
QD_Sx:	DWORD	ID_Sx:	DWORD

Index Byte	0	1	2	3	4	5	6	7		60	61	62	63
Index Word	0		1		2		3		...	30		31	
Index DW	0				1				..	15			

For SYMAC the SYNC AREA is up to DW 31

6.3.4 Access to Application Interface (API)

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

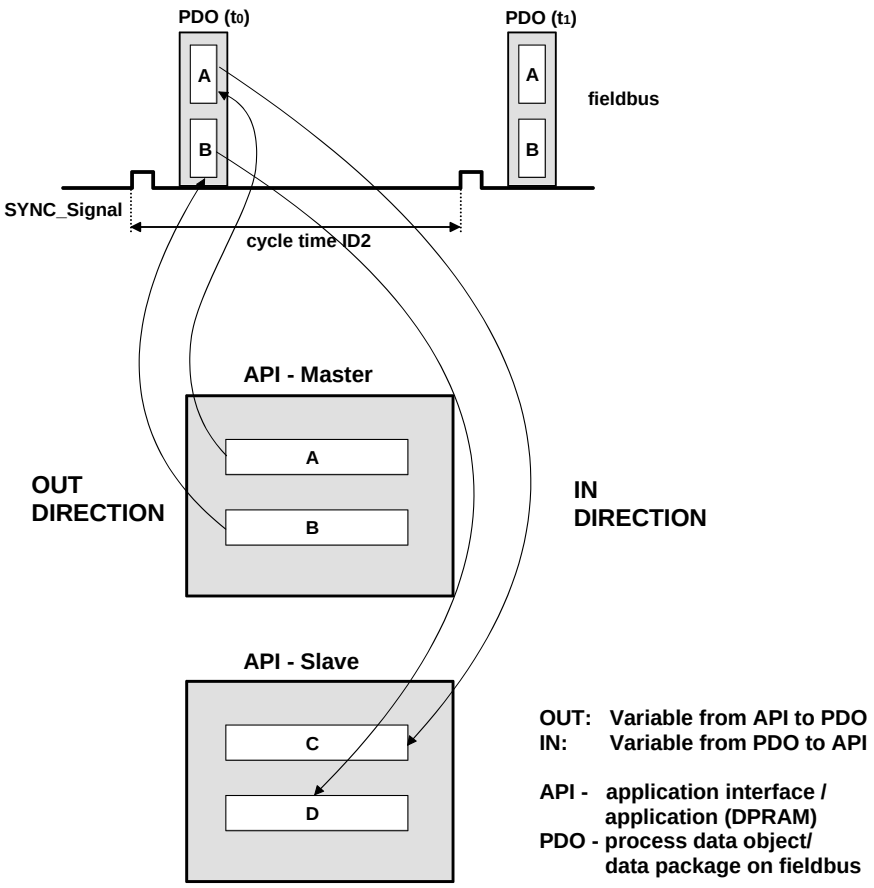


Figure 6-1: Scheme of data transfer

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.

6.3.5 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)
2	additional_set_point	0x2030	0x03	CAN to API	0.5	1)
	Actual values					
2	actual_16bit_message	0x2040	0x01	API to CAN	0.5	2)
4	actual_32bit_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	12)
2	device_control_bits	0x2049	0x00	CAN to API	5.0	13)
2	real_time_bits	0x204A	0x00	API to CAN	5.0	14)
	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)
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)

length [byte]	CAN name	CAN Index	CAN Sub index	CAN copy direction CAN<->API	Copy time [ms]	
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	Async	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)
	Setpoint Source					
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)
	AFP area					
8	AFP write				5.0	10)
8	AFP read				5.0	11)

1) function of operating mode (ID32800...)
   main_set_point : 32bit setpoint source
   additional_set_point: 16bit setpoint source
2) configurable 16bit message (ID32785)
3) configurable 32bit 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 dictionary object 0x2021 sub1 to sub8.

11) AFP read block only accessible through dictionary object 0x2020 sub1 to sub8.

12) 13)

Device Control bits		Device Status bits	
Bit	Meaning	Bit	Meaning
0	FL (error deletion)	0	SBM (group ready message)
1	UE (inverter ON)	1	QUE
2	Reserve	2	Reserve
3	Reserve	3	Reserve
4	Reserve	4	Reserve
5	Reserve	5	Reserve
6	Reserve	6	Reserve
7	Reserve	7	Reserve
8	RF1 (controller enable AW1)	8	QRF1 (acknowledgement RF AW1)
9	RF2 (controller enable AW2)	9	QRF2 (acknowledgement RF AW2)
A	RF3 (controller enable AW3)	A	QRF3 (acknowledgement RF AW3)
B	RF4 (controller enable AW4)	B	QRF4 (acknowledgement RF AW4)
C	RF5 (controller enable AW5)	C	QRF5 (acknowledgement RF AW5)
D	RF6 (controller enable AW6)	D	QRF6 (acknowledgement RF AW6)
E	RF7 (controller enable AW7)	E	QRF7 (acknowledgement RF AW7)
F	RF8 (controller enable AW8)	F	QRF8 (acknowledgement RF AW8)

14) Real Time Bits

Bit	Meaning
0	$\lvert N_{command} - N_{feedback} \rvert < Nwindow$ ($N_{feedback} = N_{command}$), according to window ID157
1	$\lvert N_{feedback} \rvert < Nmin$, according to window ID124
2	$\lvert N_{feedback} \rvert < n_x$, according to window ID125
3	$\lvert M_{feedback} \rvert > M_x$, according to window ID126
4	$\lvert M_{feedback} \rvert > M_{limit}$, according to ID82, ID83
5	$\lvert N_{command} \rvert > \lvert N_{limit} \rvert$, according to ID38, ID39
6	$SA < SA_{limit}$, according to window ID57
7	$\lvert P_{feedback} \rvert \geq P_x$, according to window ID158
8	Software limit switch negative, according to window ID50
9	Drive angle synchronous, according to window ID228
A	Drive position synchronous, according to window ID32952
B	$N_{feedback} \geq 0$, as from AW version AW 0210
C	Feedback value acknowledgement calibrated
D	Residual distance was deleted, according to window ID32922
E	Overcurrent message: Utilization > 50% of the overload limit
F	Software limit switch positive according to window ID49

Key:
N Speed, M Torque, P Power, X Position, SA Following error
–

All this functionality is valid with PLC software **AE-PLC V1.01 2002/02** and KU version **KU 1.11 3900**.

6.4 Transmission Type (alias TransTyp)

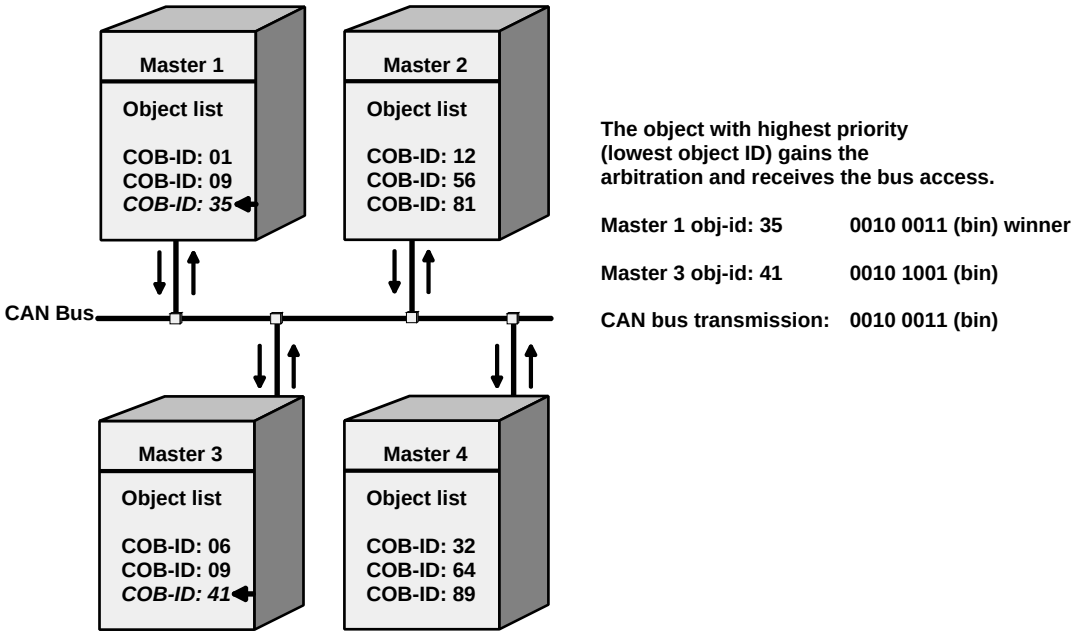
TransType	Meaning
CycSync x	Transmission rate of x means that the **synchronous PDO** message is transmitted with every x-th SYNC object. The cycle time of the sync object is ID2 SERCOS cycle time. e.g. ID2=2ms, CycSync3 : transmitted every 6ms (commands, setpoints, actual values,...)
AMKevent, DeviceEvent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

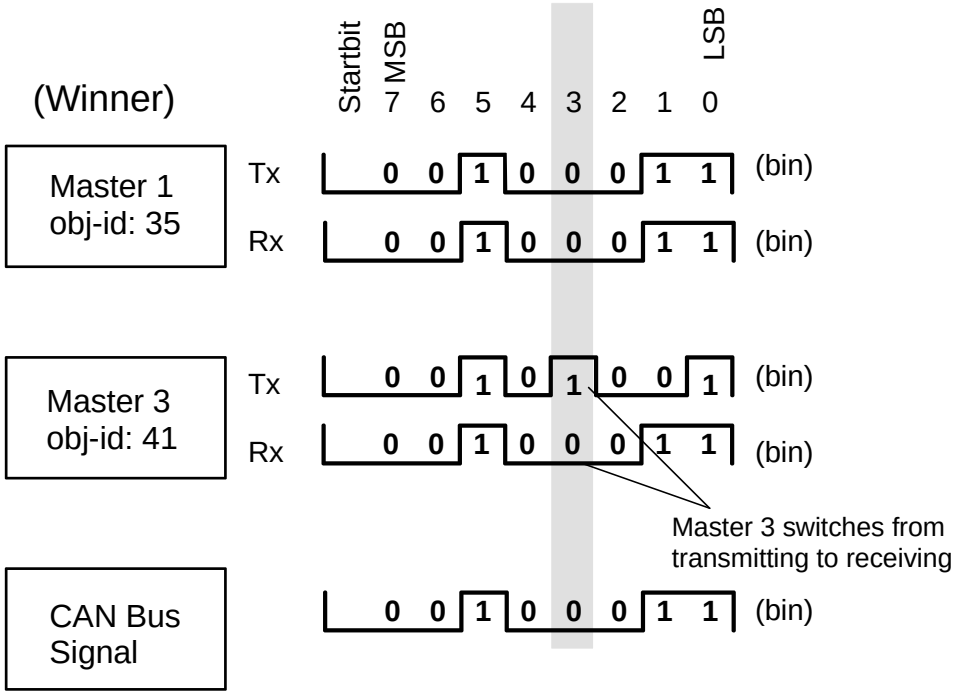
6.5 COB-ID and arbitration prinziple (alias COBpdo)

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




(Winner)
Startbit
MSB
LSB
7 6 5 4 3 2 1 0
Master 1
obj-id: 35
Tx
Rx
0 0 1 0 0 0 1 1 (bin)
Master 3
obj-id: 41
Tx
Rx
0 0 1 0 1 0 0 1 (bin)
0 0 1 0 0 0 1 1 (bin)
Master 3 switches from
transmitting to receiving
CAN Bus
Signal
0 0 1 0 0 0 1 1 (bin)

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.

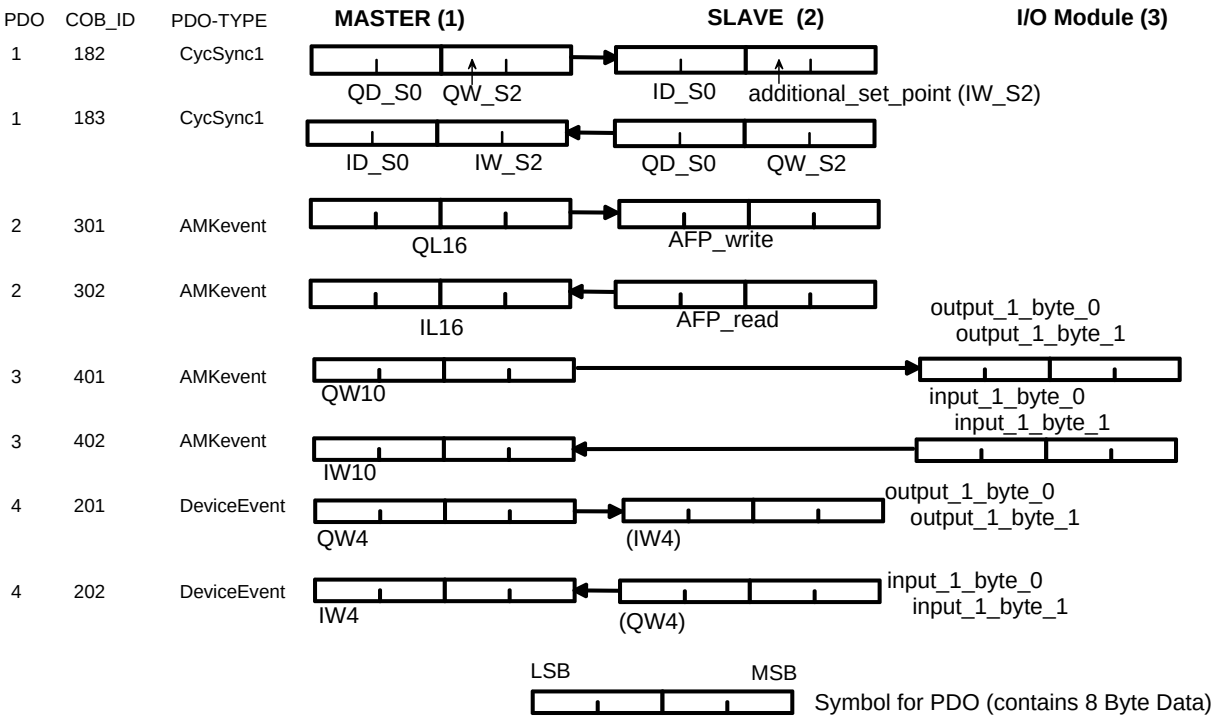


Figure 7-1: Data exchange definition

The following CCF-File describes the application according to the above definitions. Converted from *.ccf into *.ccb with the tool CANConv and the *.ccb File needs to be downloaded into the master with the tool DVLader.

```
// ===================================================================
//   Filename: can5.ccf
//   Projekt :
//   Date    : 01.02,02
//
// ===================================================================
// History
// 31.01.02
====================================================================

      readFile predef2.ccf                    //new Mapping parameter
      readFile predefAPI.ccf


//------------------------
//   Common parameters
//------------------------
      nodelist 1 2 3
      nodegroup slaves 2 3

      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 Adr. 1
//--------------------------------------

      alias master       1


//   1.Transmit PDO       -> Slave Adr.2
      alias PDOno        1
      alias Map          QD_S0 QW_S2    //QW_S2 -> additional Setpoint
      alias TransTyp     CycSync1
      alias COBpdo       0x00000182
      aliasend

      confMasterTransmitPDO


//   2.Transmit PDO       AFP -> Slave Adr.2
      alias PDOno        2
      alias Map          QL16     //AFP_write
      alias TransTyp     AMKevent
      alias COBpdo       0x00000301
      aliasend

      confMasterTransmitPDO

//   3.Transmit PDO       I/O -> Slave Adr.3          WAGO EA 16
      alias PDOno        3
      alias Map          QW10
      alias TransTyp     AMKevent
      alias COBpdo       0x00000401
      aliasend

      confMasterTransmitPDO

//   4.Transmit PDO       I/O EA1-> Slave Adr.2
      alias PDOno        4
      alias Map          QW4
      alias TransTyp     AMKevent
      alias COBpdo       0x00000201
      aliasend

      confMasterTransmitPDO
```

```
//  5. Receive PDO        <- Slave Adr.2
      alias PDOno        1
      alias Map          ID_S0 IW_S2
      alias TransTyp     CycSync1
      alias COBpdo       0x00000183
      aliasend

      confMasterReceivePDO


//  6. Receive PDO        AFP <- Slave Adr.2
      alias PDOno        3
      alias Map          IL16
      alias TransTyp     AMKevent
      alias COBpdo       0x00000302
      aliasend

      confMasterReceivePDO


//  7. Receive PDO        I/O <- Slave Adr.3          WAGO EA16
      alias PDOno        4
      alias Map          IW10
      alias TransTyp     AMKevent
      alias COBpdo       0x00000402
      aliasend

      confMasterReceivePDO


//  8. Receive PDO        I/O <- Slave Adr.2
      alias PDOno        2
      alias Map          IW4
      alias TransTyp     AMKevent
      alias COBpdo       0x00000202
      aliasend

      confMasterReceivePDO


//-------------------------
//   Slave 1 Adr.2
//-------------------------

     alias slave        2

//   1.Transmit PDO       -> Master Adr.1
      alias PDOno        1
      alias Map          QD_S0 QW_S2
      alias TransTyp     CycSync1
      alias COBpdo       0x00000183
      aliasend

      confSlaveTransmitPDO


//   2.Transmit PDO       -> Master Adr.1
      alias PDOno        2
      alias Map          input_1_byte_0 input_1_byte_1 //QW4
      alias TransTyp     AMKevent
      alias COBpdo       0x00000202
      aliasend

      confSlaveTransmitPDO


//   3.Transmit PDO       AFP -> Master Adr.1
      alias PDOno        3
      alias Map          AFP_read
      alias TransTyp     AMKevent
      alias COBpdo       0x00000302
      aliasend

      confSlaveTransmitPDO
```

```
//   4.Receive PDO        <- Master Adr.1
      alias PDOno        1
      alias Map          ID_S0 additional_set_point  //IW_S2
      alias TransTyp     CycSync1
      alias COBpdo       0x00000182
      aliasend

      confSlaveReceivePDO


//   5.Receive PDO        I/O EA1 <- Master Adr.1
      alias PDOno        2
      alias Map          output_1_byte_0 output_1_byte_1      //IW4
      alias TransTyp     AMKevent
      alias COBpdo       0x00000201
      aliasend

      confSlaveReceivePDO


//   6.Receive PDO        AFP <- Master Adr.1
      alias PDOno        3
      alias Map          AFP_write
      alias TransTyp     AMKevent
      alias COBpdo       0x00000301
      aliasend

      confSlaveReceivePDO



//---------------------------
//   Slave 3 Configuration WAGO EA16
//---------------------------
            alias slave            3

//  1. Transmit PDO to Master
      alias PDOno        1
      alias Map          input_1_byte_0  input_1_byte_1
      alias TransType    DeviceEvent
      alias COBpdo       0x00000402
      aliasend

      confSlaveTransmitPDOnew

//  2. Receive PDO  from Master
      alias PDOno        2
      alias Map          output_1_byte_0  output_1_byte_1
      alias TransTyp     DeviceEvent
      alias COBpdo       0x00000401
      aliasend

      confSlaveReceivePDOnew


      readFile confCommon.ccf

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