SYS TEC ELECTRONIC

CANopen Bootloader

Software Manual

Edition August 2008

The designations used in this book for products that are also registered trademarks have not been separately marked. The missing © sign does not therefore imply that the designation is a non-registered product name. At the same time, no inference can be made from the designation used that this refers to any pending patent or that a registered design is protected.

The information in this manual has been carefully checked and can be assumed to be correct and accurate. However, we expressly draw your attention to the fact that the SYS TEC electronic GmbH company neither excepts warranty claims nor legal responsibility, nor claims of any sort from secondary damages resulting from the use of this manual or its content. We reserve the right to change the details specified in this manual without notice. The SYS TEC electronic GmbH company does not thereby incur any obligations.

Furthermore, we also expressly draw your attention to the fact that the SYS TEC electronic GmbH company neither excepts warranty claims nor legal responsibility, nor claims of any sort from secondary damages resulting from the incorrect use or incorrect application of the hardware or software. The layout and/or design of the hardware may also be changed without notice. SYS TEC electronic GmbH does not thereby incur any obligations.

	EUROPE	NORTH AMERICA
Address:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Sales hotline:	+49 (0) 36 61 / 62 79-0 info@systec-electronic.com	+1 (800) 278-9913 order@phytec.com
Technical hotline:	+49 (0) 36 61 / 62 79-0 support@systec-electronic.com	+1 (800) 278-9913 support@phytec.com
Fax:	+49 (0) 36 61 / 6 79 99	+1 (206) 780-9135
Web page:	http://www.systec-electronic.com	http://www.phytec.com

5th Edition August 2008

Table of contents

1 Introduction 1
2 References 2
3 Design of data transmission 3
3.1 Checksum application 11
3.2 Starting the bootloader 12
3.3 Software structure of the target 14
4 Interfaces 17
4.1 Interface to the flash 17
4.1.1 Interface to the timer 22
4.1.2 Interface for NodeID 25
4.1.3 Application parameter 26
4.1.3.1 TgtGetAppInfo 26
4.1.3.2 TgtGetAppSize 26
4.1.3.3 TgtSetAppSize 27
4.1.3.4 TgtGetAppCrc 27
4.1.3.5 TgtSetAppCrc 28
4.1.3.6 TgtCheckAppSig 28
4.1.3.7 TgtClrAppSig 29
4.1.3.8 TgtSetAppSig 29
4.1.3.9 TgtGetAppSig 29
4.1.3.10 TgtGetSerialNr 29
4.1.4 Re-entry in the bootloader 29
4.1.5 Interface to the CAN bus 30
4.1.6 Debug outputs 30
4.2 Configuring the bootloader 31
5 Flash tool 35
5.1 BinaryBlock-Converter 35
5.2 BinaryBlock-Download 38
5.3 Configuring the flash tool software (FtCfg.h) 40
5.4 Error codes of the flash tool software (FtErrDef.h) 42
6 Resources 45
6.1 Code, data target 45
6.2 Target interrupts 45

Table of figures

Figure 1: CANopen communication via objects 5
Figure 2: Data format of a block with programme data 8
Figure 3: Sequence of data transmission 9
Figure 4: Construction of the flash 11
Figure 5: Programme sequence when starting the bootloader..... 12
Figure 6: Software structure of the bootloader 14

List of tables

Table 1: List of CANopen objects for the bootloader..... 4
Table 2: Bootloader commands in index 0x1F51 6
Table 3: State of the programme download in index 0x1F57..... 7
Table 4: Source files for the bootloader 15
Table 5: Meaning of the flash tool error codes 44

1 Introduction

The CANopen Bootloader is a software package used to transfer programs in binary format to the target hardware and to run them there using CANopen. The functionality of the programme is geared to the determinations set down in the CANopen Standard CiA DS-302.

The software package is comprised of two parts: the flash tools and the bootloader. The flash tools convert the application data (S3, INTEL-Hex) into a binary format and transfer them to the target hardware. The bootloader receives the data transmitted by the flash tools, verifies them and writes the data into the flash; it then starts the application that has been transferred.

Communication and data transmission between the bootloader and the flash tools is by means of CANopen SDO transfer.

This manual describes the mode of operation of the bootloader, the format of the binary data and the interface to adapt the bootloader to the user hardware.

- Interface to flash (section)
- Interface to system timer (section)
- NodeID interface if this can not be permanently configured (section.)
- Interface to re-enter the bootloader from the application (section)
- Interface to the CAN-Bus if no CAN driver exists for the selected CAN controller (section)
- Interface to the debug outputs (section)
- Interrupt vector table interface (section)

The interface to the user hardware has been implemented as a template.

2 References

/1/ Framework of CANopen Manager, CiA DS-302, 2006

3 Design of data transmission

Implementation is based on the CANopen Standard CiA DS-302. This standard defines object entries that can be used to download a programme. The bootloader uses service data objects (SDO) for data transmission. Depending on the application of the bootloader, the loaded data can be programmed into a non-volatile flash memory.

The advantage of the use of CANopen SDO transfer to transmit data is that CANopen tools can be used to programme the application. An EDS file is required for this that maps the object entries of the CANopen nodes. The data is transmitted in binary format and must be converted into this format (e.g. HEX → BIN) depending on the development environment.

The bootloader is a complete CANopen application and it therefore has an object directory. However, this object directory is only visible for the time in which the bootloader is run.

The bootloader is comprised of a target-related part and a part that packages the communication services. The target-specific part contains the interfaces to the flash, the CAN controller and the application. Adjustments must be made by the user here.

SDO transfer according to CiA DS-301 is used for data transfer. This transmission service supports protocols such as segmented transfer and block transfer. The default setting is segmented transfer because this service requires less CODE to be run. However, due to the acknowledgment sequence of the data, which is segment-by-segment, the transmission time may be slightly longer in comparison to block transfer.

Object (Index/ Subidx)	Data type	Attribute	Meaning	CiA Stan.
0x1000	Unsigned32	ro	Designates the device type	301
0x1001	Unsigned8	ro	CANopen error register	301
0x1018/0	Unsigned8	ro	Number of sub-indices in 0x1018	301
0x1018/1	Unsigned32	ro	CANopen vendor ID	301
0x1018/2	Unsigned32	ro	CANopen product ID	301
0x1018/3	Unsigned32	ro	Serial number	301
0x1018/4	Unsigned32	ro	Revision number	301
0x1F50/0	Unsigned8	ro	Number of sub-indices in 0x1F50	302
0x1F50/1	Domain	wo	Programme data (container up to 16384 bytes)	302
0x1F51/0	Unsigned8	ro	Number of sub-indices in 0x1F51	302
0x1F51/1	Unsigned8	rw	Command channel	302
0x1F56/0	Unsigned8	ro	Number of sub-indices in 0x1F56	302
0x1F56/1	Unsigned32	ro	Identifies the application (CRC of the application)	302
0x1F57/0	Unsigned8	ro	Number of sub-indices in 0x1F57	302
0x1F57/1	Unsigned32	ro	Flash error status	302

Table 1: List of CANopen objects for the bootloader

The 0x1F50 object is used to load the programme data. The object-type is DOMAIN and it can receive data up to a block size defined by the user. The 0x1F51/1 object is used to run certain commands (deleting the flash, starting the application, etc.). The 0x1F56 object contains an identification of the application. The CRC of the application is stored here. The current error status can be read out by object 0x1F57. ***Table 1*** lists all CANopen objects and their meaning.

Meaning of the sub-index

The CANopen standard defines up to 255 sub-indices for each index. Sub-index 0 hereby contains the number of the subsequent sub-indices. A program is assigned to each sub-index for the bootloader functionality, i.e. it is possible to transmit up to 255 different programmes within an

application. Each programme must therefore be assigned an area within the flash for storage. All instructions, such as delete, write, or programme for a selected programme must be transmitted to this area with the same reference, i.e. the same sub-index.
This implementation supports one programme and therefore only one sub-index.

Figure 1 shows a diagram of communication via CANopen objects using the SDO service.

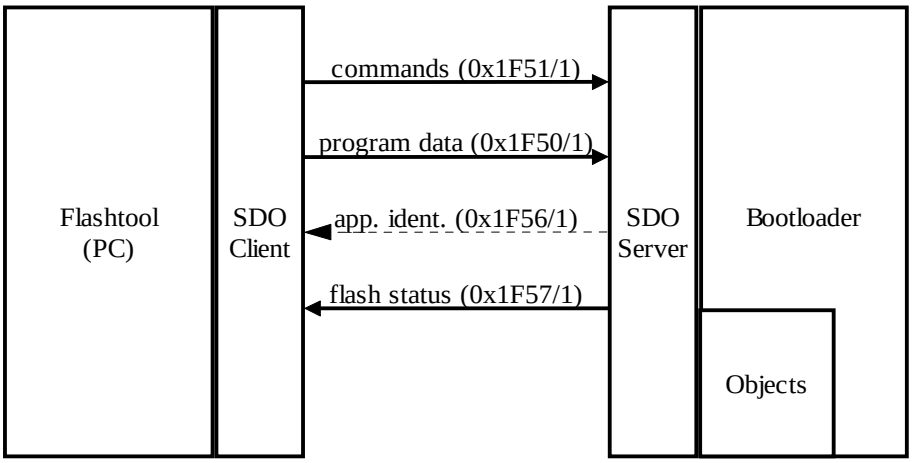

Figure 1: CANopen communication via objects

Meaning of the commands
The bootloader on the target has a command interface. Normally it only runs instructions requested by the host. The host transmits commands (writing a command to object 0x1F51 or transferring a block of data to object 0x1F50) and checks execution status using object 0x1F57.

The following commands can be run via index 0x1F51:

Value	Command	Meaning
0x00	STOP	The target is instructed to stop the running programme. This command is currently not implemented.
0x01	START	The target is instructed to start the selected programme.
0x02	RESET_STAT	The target is instructed to reset the status (Index 0x1F57).
0x03	CLEAR	The target is instructed to clear that area of the flash that has been selected with the appropriate sub-index.
0x80	START BOOTLOADER	You can jump back from the application into the bootloader using this command. This entry must therefore also be supported by the application to start the bootloader *(refer also to 3.2).*
0x83	SET_SIGNATURE	The target is instructed to flag the selected and programmed program as „valid“. In addition to a valid CRC and node number, this is the requirement to start the program automatically after a power-on-reset.
0x84	CLR_SIGNATURE	The target is instructed to flag the selected and programmed program as „invalid“. After a power-on RESET, the application would then not be started; the bootloader remains active.

Table 2: Bootloader commands in index 0x1F51

The status that can be read back in index 0x1F57 can assume the following states:

Value of state	Designation	Meaning
0x00000000	OK	The last command transmitted has been run without error.
0x00000001	BUSY	A command is still being run.
0x00000002	NOVALPROG	An attempt has been made to start an invalid application programme.
0x00000004	FORMAT	The format of binary data that have been transferred to index 0x1F51 is incorrect.
0x00000006	CRC	The CRC of the binary data is incorrect.
0x00000008	NOTCLEARED	An attempt has been made to programme although there is a valid application programme.
0x0000000A	WRITE	An error occurred during the writing of the flash.
0x0000000C	ADDRESS	An attempt has been made to write an invalid address into the flash.
0x0000000E	SECURED	An attempt has been made to write to a protected flash area.
0x00000010	NVDATA	An error has occurred when accessing the non-volatile memory (e.g. programming the signature).

Table 3: State of the programme download in index 0x1F57

Downloading the data

The binary data of a programme is transmitted in blocks. After every successful transmission of a block, this is written into the flash. ***Figure 2*** shows how a block is constructed.

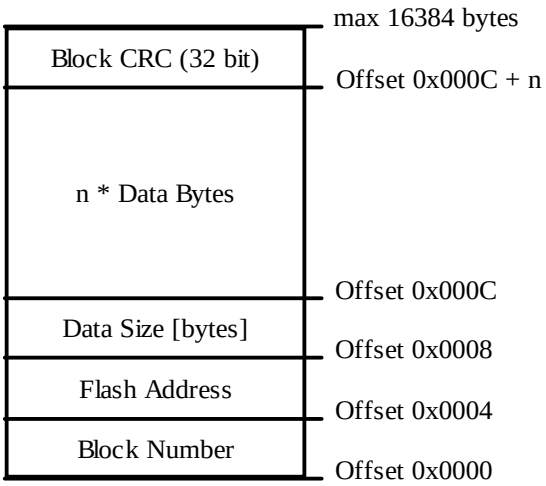

Figure 2: Data format of a block with programme data

A CAN message is already stored by the CAN controller with a CRC. A CRC is additionally appended to each block. It is calculated based on the block number, the flash address, the number of data bytes and the data bytes themselves. A block that is transmitted incorrectly is repeated n times (parameter PC tool).

Important:
Data is always transmitted in Intel format (Little Endian / LSB first). This must be considered when creating a block.

The block number is a consecutive number starting with 0; this is increased by 1 after every successful transmission. Block number 0 thereby always designates the start of transmission. The host side and the bootloader are thereby synchronised to the target. Block 0 contains control information for the bootloader and the target (block size, flash information if required) and not yet any programme data. Programme data is then transmitted from block number 1. The last block always contains block number „-1“ (0xFFFFFFFF). The application size and the application CRC is entered in this block. After transmission of this block, the bootloader starts to generate the CRC via the application in order to compare it. The result is

then also entered into object 0x1F56/1. The flash tool only starts the application after this by entering the „Start“ command into object 0x1F51/1.

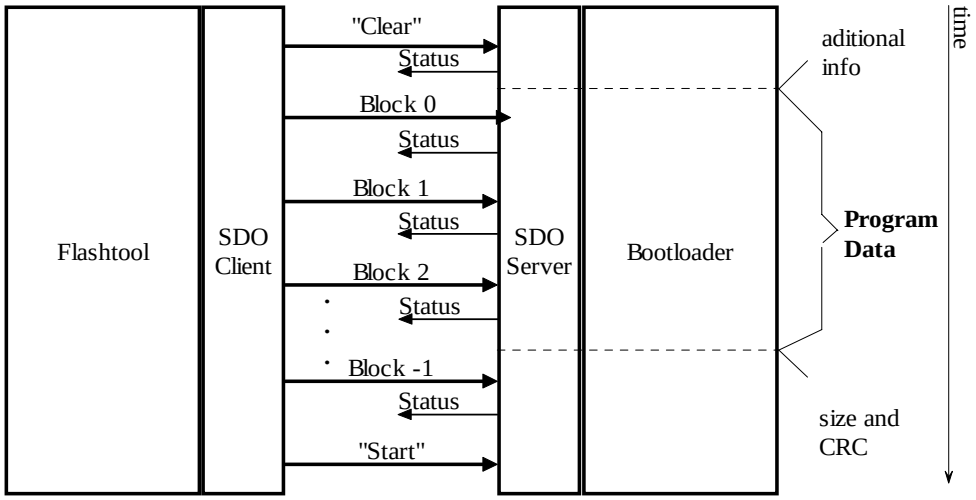

Figure 3: Sequence of data transmission

Procedure in principle

To transfer a new application to the target, the host (flashtools) carries out the following procedure. In case errors occur, the whole procedure is stopped.

1. Transforming the application into binary format (e.g. HEX -> BIN).
2. Reading object 0x1000 (devicetype) to observe if the bootloader is active. If the value detected equals the one of the bootloader (0x10000000), it is to be proceeded with step 4. In case of faulty SDO-transfer with timeout, this step is to be repeated.
3. Running the command to return to the bootloader (writing the value 0x80 to object 0x1F51/1). Afterwards step 2 is to be repeated.
4. Running the command to erase the flash (writing the value 0x03 to object 0x1F51/1).
5. Reading object 0x1F51 to observe if the erasure has been carried out. In case of faulty SDO-transfer with timeout or if the value BUSY was delivered back, this step is to be repeated.
6. Loading the first block of data from the binary data file.
7. Downloading the block of data to object 0x1F51/1.

8. Reading object 0x1F51 to observe if the writing of the blocks of data have been carried out. In case of faulty SDO-transfer or if the value BUSY was delivered back, this step is to be repeated.
9. If another block of data in binary format is available, read it and proceed with step 7.
10. Running the command Starting the application (writing the value 0x01 to object 0x1F51/1).
11. Reading of object 0x1000 (devicetype) to observe if the bootloader is still active. If the value detected equals the one of the bootloader (0x10000000), the whole procedure is to be stopped with an error. In case of faulty SDO-transfer with timeout, this step is to be repeated.
12. Running the command to return to the bootloader (writing the value 0x80 to object 0x1F51/1).
13. Reading of object 0x1000 (devicetype) to observe if the bootloader is active. If the value detected does not equal the one of the bootloader (0x10000000), the whole procedure is to be stopped with an error. In case of faulty SDO-transfer with timeout, this step is to be repeated.
14. Running the command to place the signature (writing the value 0x83 to object 0x1F51/1).
15. Running the command Starting the application (writing the value 0x01 to object 0x1F51/1).

Timeouts

Two timeouts are relevant for transmitting the data from the host (flash tool) to the target (bootloader). One of these is the timeout for acknowledgement of the transmitted SDO service (an SDO transfer is always acknowledged). This timeout is relatively short as only the delay caused by the transmission path (bit rate, load on bus) must be considered here.

The other timeout must be selected depending on the command transmitted. In general, a command must be completely transmitted and acknowledged before being run. The execution time, and therefore the timeout to be selected at the host, depends on the command in question. If necessary, other times must be considered for deleting the flash or writing the data than for writing a signature. The user must configure the correct timeouts.

3.1 Checksum application

A checksum is generated for each data block as well as for the entire application area. It is assumed that the area for the application can be mapped linearly into the address area of the CPU.

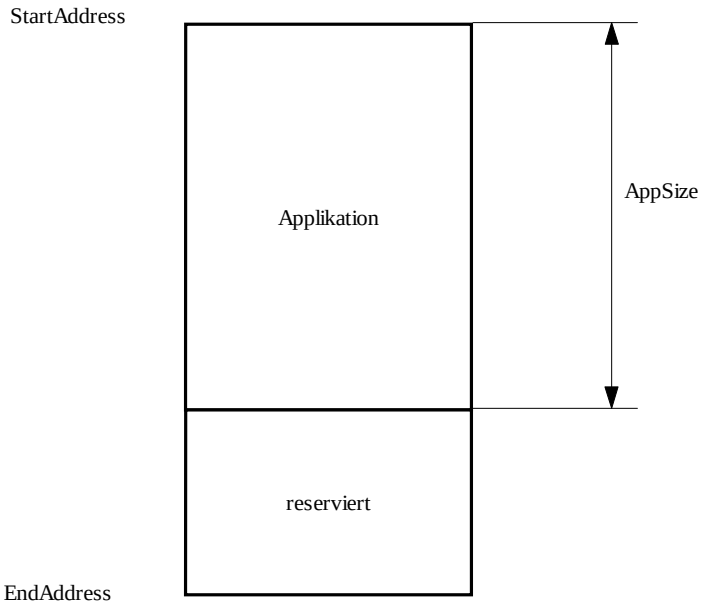

Figure 4: Construction of the flash

The start address is defined as a constant in the source. The CRC as well as the size of the application are transmitted by the host to the target in the last block. The target stores these two values into a non-volatile memory. There are target-specific templates for this; these must be adapted by the user. The flash itself or an EEPROM can be used as non-volatile memory.
The CRC is calculated via the application (length of AppSize) of the application area (for implementation, please refer to Crc32.c). The start value is 0. StartAddress and EndAddress are stored as constants and must be identical to the parameters used in creating the binary data.

3.2 Starting the bootloader

The ***Figure 5*** shows the sequence implemented when starting the bootloader.

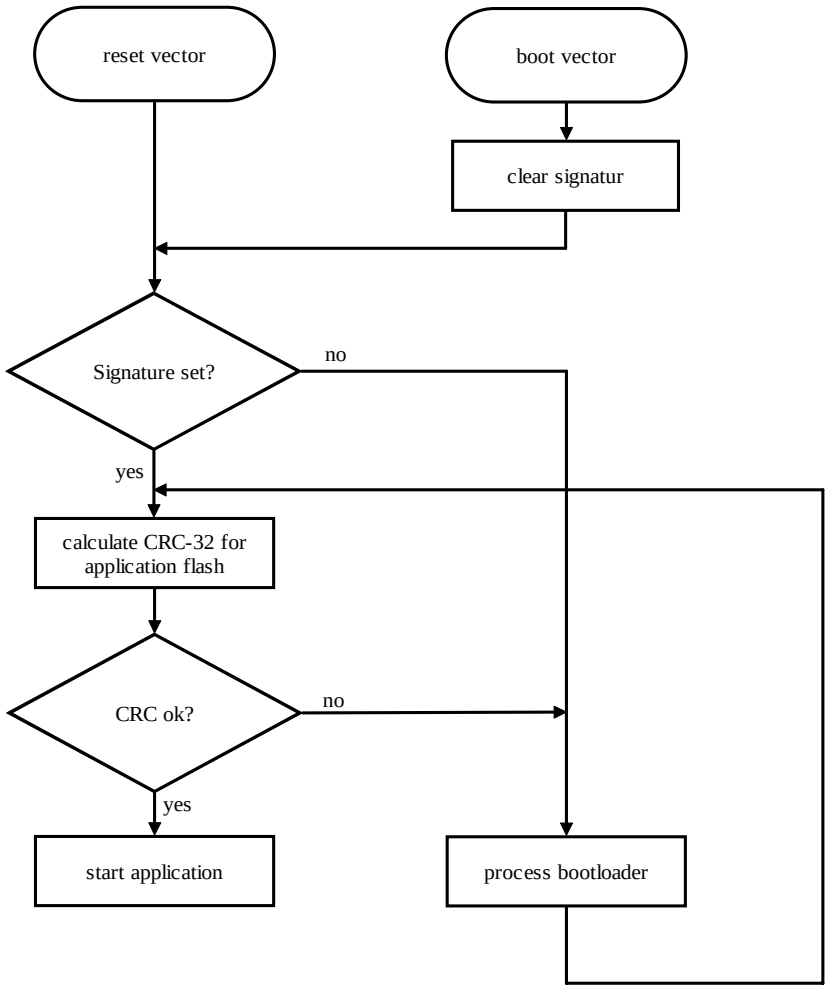

Figure 5: Programme sequence when starting the bootloader

There are two entry points for the bootloader. After the hardware reset, the CPU starts up the reset vector. The second entry point is an entry vector provided by the bootloader to allow the application to activate the bootloader. Entry via the boot vector always starts the bootloader independent of the state of the application.

Starting the application presupposes that the CRC via the application area is identical to the CRC calculated on the host and that is stored on the target,

that there is a valid node number (the node numbers must be within a value area defined for this application) and that a valid signature has been stored. If one of these conditions is not fulfilled then the target dwells in the bootloader.

There are various ways of implementing the signature. The signature can be stored in the flash or in another non-volatile memory (e.g. EEPROM, NVRAM).
However, it is also possible to implement the signature using a port pin. In all cases, if the signature is not set then the bootloader is started.

Note: There are various ways of implementing a re-entry from the application into the bootloader. On the one hand, this depends on how a re-entry triggered by the microcontroller itself, as well as by the compiler and linker, are supported. Observe in this regard that when closing the application, all of the microprocessors resources such as interrupts, DMA channel, and on-chip periphery are released (block interrupts, end DMA transfer, block periphery) and the system stack is reset. A simple way of doing this is the method shown here - by triggering a RESET. However, to do this, the signature in the non-volatile memory must be deleted immediately beforehand. When starting the target after a RESET, one of the conditions for starting the application is not fulfilled and the target therefore dwells in the bootloader and waits for additional commands from the host.
To implement a start of the bootloader via the OD, index 0x1F51 – „Program Control“ and the RESET command *(refer to Table 2)* must be implemented in the application.

3.3 Software structure of the target

Figure 6 shows the software structure of the bootloader.

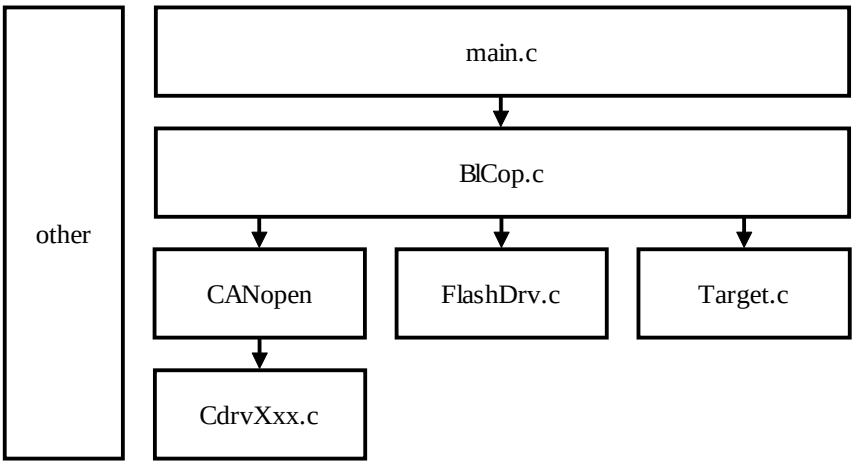

Figure 6: Software structure of the bootloader

File	Meaning
Directory of CANopen_Bootloader	
Source\main.c	Contains the main() function for the bootloader
Source\BlCop.c	Contains the interface functions of the bootloader
Source\Crc32.c	Contains functions for calculating a 16 bit CRC
Objdicts\Objdict.*	Definition of the object directory, EDS file of the bootloader
Target\XC16x\Source\FlashDrv.c	Contains functions to delete and write the flash and must be adjusted to the target accordingly
Target\XC16x\Source\Target.c	Contains the target-specific functions for deleting and writing the signature, CRC, application size, etc.
Target\XXX\Include\BlCopCfg.h	Constants for bootloader configuration
Target\XXX\Include\CopCfg.h	Constants for CANopen configuration
Target\XXX\Include\CdrvTgt.h	Definition of the CAN driver
Directory of CANopen_V5	
CopStack\SdosComm.c	Functions for the SDO server
CopStack\Obd.c	Functions for access to the object directory
CopStack\Cob.c	Functions used for generating the communication objects.
CopStack\AmiXXX.c	Contains special functions for memory access
CopStack\LSSS.c	Only when supported by LSS
Cdrv\CdrvXXX.c	Contains driver functions for access to the CAN controller
Cdrv\BdiTabXX.c [1]	Contains a baud rate table for the CAN interface

Table 4: Source files for the bootloader

4 Interfaces

4.1 Interface to the flash

The following functions must be implemented. During in-house integration at SYS TEC electronic, perfect function must be proven by a demonstration programme.

a) FlsDrvInitialize function

```
tFlsDrvResult FlsDrvInitialize (void);
```

Meaning:
This function initialises the flash driver. If the internal functions can only be run from the RAM then the required functions are copied into the RAM. Access to these functions is then implemented by function pointers. The function call must be implemented from within the flash driver. It is the responsibility of the user to initialise the function pointer.

Parameters:
none

Response:
This function returns a 0 if initialisation was successful.

b) FlsDrvEraseInitialize function

```
tFlsDrvResult FlsDrvEraseInitialize (DWORD dwStartAddr_p, DWORD
dwEndAddr_p);
```

Meaning:
This function initialises the deletion of an application area. An application area is designated by its start address and its end address and can be comprised of one or more flash sectors/page. The start address must be lower than the end address.

Parameters:
dwStartAddr_p: Start address of the application area
dwEndAddr_p: End address of the application area

Response:
This function returns a 0 if initialisation was successful.

c) FlsDrvEraseSector function

```
tFlsDrvResult FlsDrvEraseSector (DWORD dwSecAddr_p, DWORD *
pdwSize_p);
```

Meaning:
This function deletes a part of the flash within the application area (the application area was specified by the FlsDrvEraseInitialize function using the dwStartAddr_p, dwEndAddr_p parameter). An application area can be comprised of several flash sectors/pages. The value dwSecAddr_p zeigt points to the first cell to be deleted within the area. The pdwSize_p pointer points to the size of the area to be deleted. The number of bytes to be actually deleted is stored in *pdwSize_p.

During deletion, the FlsDrvResetWatchdog function may have to be run in cycles to reset the watchdog.

Parameters:
dwSecAddr_p: Start address of the area to be deleted
*dwSize_p: Pointer to the size of the area

Response:
This function returns a 0 if deletion of the flash sector was successful. The dwSize_p pointer points to the number of bytes actually deleted.

d) FlsDrvWriteData function

```
tFlsDrvResult FlsDrvWriteData (BYTE* pbData_p,
    DWORD dwStart_p, DWORD dwSize_p);
```

Meaning:
This function writes data in the flash area of the application. If required, this function also cyclically calls the FlsDrvResetWatchdog function.

Parameters:

pbData_p:	This pointer indicates the start address of the source data to be written to the flash.
dwStart_p:	This parameter indicates the start address of the flash to which the source data is to be written.
dwSize_p:	This parameter indicates the number of bytes to be written into the flash.

Response:
This function returns a 0 if writing to the flash was successful.

e) FlsDrvVerifyData function

```
tFlsDrvResult FlsDrvVerifyData (BYTE* pbData_p,
    DWORD dwStart_p, DWORD dwSize_p);
```

Meaning:
This data checks the data in the flash area against the transmitted data. The function cyclically calls the FlsDrvResetWatchdog function.

Parameters:

pbData_p:	This pointer indicates the start address of the source data to be checked against the data in the flash.
dwStart_p:	This parameter indicates the start address of the flash from where the data

	is to be read.
dwSize_p:	This parameter indicates the number of bytes to be checked in the flash.

Response:
This function returns a 0 if checking of the flash was successful.

f) FlsDrvReadData function

```
tFlsDrvResult    FlsDrvReadData    (BYTE**    pbData_p,    DWORD
*pdwStartAddr_p, DWORD *pdwSize_p);
```

Meaning:
This function reads out a data block. The start address is thereby incremented by the read number of bytes. The pointer *pdwSize_p points to the number of read bytes when quitting the function.
The function packages special features target-specific when reading the code memory.

Parameters:

pbData_p:	Pointer to the address with the read data
pdwStartAddr_p:	Pointer to the start address of the block to be read in the code memory
pdwSize_p:	Pointer to the number of bytes

Response:
This function returns a 0 if deletion of the flash sector was successful. Additionally, the start address of the next block is calculated (pdwStartAddr_p) by the function and the actual number of read bytes is calculated (pdwSize_p).

g) FlsDrvEnterCriticalSection function

```
void FlsDrvEnterCriticalSection (void);
```

Meaning:
This function blocks the global interrupt flag during the time in which the flash is accessed. This function is required within the flash driver when running of machine commands is not possible during deletion or when programming data.

Parameters:
none

Response:
none

h) FlsDrvLeaveCriticalSection function

```
void FlsDrvLeaveCriticalSection (void);
```

Meaning:
This function once more releases the global interrupt flag.

Parameters:
none

Response:
none

i) FlsDrvResetWatchdog function

```
void FlsDrvResetWatchdog (void);
```

Meaning:
This function resets the watchdog. This function is required when the execution period of an internal function nears the watchdog timeout.

Parameters:
none

Response:
none

4.1.1 Interface to the timer

The following functions must be implemented. If possible, the use of interrupts should be avoided. The system time is required for monitoring timeouts during an active SDO connection. As these times are processed within CANopen as multiples of 100µs, the value of the timer must be sized accordingly, even if the accuracy is only a value of 1ms.

a) TgtInitTimer

```
void TgtInitTimer (void);
```

Meaning:
This function initialises a system timer for provision of a system timing clock of 1ms.

Parameters:
none

Response:
none

b) TgtStopTimer

```
void TgtStopTimer (void);
```

Meaning:
This function stops the system timer and once more releases the used resources (timer, interrupt).

Parameters:
none

Response:
none

c) TgtTimerIsrHandler

```
void TgtTimerIsrHandler (void);
```

Meaning:
This function is called during an interrupt of the system timer in order to increment the tick count.

Parameters:
none

Response:
none

d) TgtGetTickCount

```
tTime TgtGetTickCount (void);
```

Meaning:
This function returns the current value of the system timer, whereby the return value is sized to a multiple of 100µs.

Parameters:
none

Response:
TickCount * 10 (corresponds to 100µs)

4.1.2 Interface for NodeID

This interface is required to provide a node number for the CANopen device. When configuring the node number using LSS, access to a non-volatile memory must be implemented via this interface by the user.

a) TgtGetNodeId

```
BYTE TgtGetNodeId (BYTE * pbNodeId_p);
```

Meaning:
This function returns the NodeId. This may have been set using decode switches or it is stored permanently in the firmware. When configuring the NodeId using LSS, this value must be read-out from a non-volatile memory. To start the LSS service, the function 0xFF must be returned (memory is invalid → start configuration).

Parameters:
pbNodeId_p: Pointer to NodeID

Response:
0: *pbNodeId_p = 1 .. 127, 255
>0: *pbNodeId_p = invalid

b) TgtSetNodeId

```
BYTE TgtSetNodeId (BYTE bNodeId_p);
```

Meaning:
This function stores a node number configured with LSS in a non-volatile memory.

Parameters:
Node-ID = 1 ... 127, 255

Response:
0: no error, storage successful
>0: Error code

4.1.3 Application parameter

4.1.3.1 TgtGetAppInfo

4.1.3.2 TgtGetAppSize

```
void TgtGetAppSize(DWORD GENERIC* pdwSize_p);
```

Meaning:
The TgtGetAppSize function reads the application size from the non-volatile memory of the target.

Parameters:
pdwSize_p: Pointer to the variable for storing the application size

Response:
This function returns the size of the application if the value is valid.

4.1.3.3 TgtSetAppSize

```
void TgtSetAppSize(DWORD dwSize_p);
```

Meaning:
The TgtSetAppSize function is used to store the size of the application in a non-volatile area of the target. The value is required to calculate the CRC for the application area before starting the application.

Parameters:
dwSize_p: Size of application for storage

Response:
none

4.1.3.4 TgtGetAppCrc

```
void TgtGetAppCrc(DWORD GENERIC *pdwCrc_p);
```

Meaning:
The function reads the CRC of the application from the non-volatile memory of the target. The value has been calculated on the host side and stored on programming the application using the TgtSetAppCrc function. The value is required to compare the CRC determined with the value at the host side in order to start the application.

Parameters:
pdwCrc_p: Pointer to the variable for storing the CRC

Response:
This function returns the CRC calculated at the host node and stored at the target.

4.1.3.5 TgtSetAppCrc

```
void TgtSetAppCrc(DWORD dwCrc_p);
```

Meaning:
This function stores the CRC in the non-volatile memory of the target. The CRC was calculated on the host side and transmitted to the target during the download. The CRC must be stored there non-volatile for later verification when starting the target.

Parameters:

dwCrc_p:	CRC for storage

Response:
none

4.1.3.6 TgtCheckAppSig

```
BYTE TgtCheckAppSig(void);
```

Meaning:
The signature declares the state of the application. A valid signature is the requirement to start the application. The signature is set on request from the host after the application has been verified. The function checks whether a valid signature has been stored.

Parameters:
none

Response:

TRUE:	The signature is valid
FALSE:	The signature is invalid

4.1.3.7 TgtClrAppSig

4.1.3.8 TgtSetAppSig

4.1.3.9 TgtGetAppSig

```
static BYTE TgtGetAppSig(BYTE GENERIC *pbAppSig_p);
```

Meaning:

4.1.3.10 TgtGetSerialNr

The serial number is required in order to uniquely identify the target within a CANopen network in connection with VendorID, ProductCode and RevisionCode. The functions package the access to the non-volatile memory of the target for reading or writing the serial number.
Storage of the serial number in non-volatile memory is not executed by the bootloader. It is up to the user to implement the required steps.

4.1.4 Re-entry in the bootloader

TgtJumpBootloader

```
void TgtJumpBootloader (void);
```

Meaning:
This function must be linked to a fixed address to implement a re-entry to the bootloader from the application. Before re-entry, all resources must be released by the application and the global interrupt must be blocked.

Parameters:
none

Response:
none

4.1.5 Interface to the CAN bus

This interface is based on the SYS TEC CAN drivers for CANopen. The following CAN controllers are currently supported:

Provider	CAN Controller
Infineon C16x, XC16x	TWIN-CAN MultiCAN
Freescale Coldfire, PowerPC	FlexCAN TouCAN
ATMEL ARM	AT91SAM7A3
Microchip dsPIC33F	ECAN
NXP	SJA1000 PeliCAN
Intel	82527
Renesas	M16C family
Fujitsu	16LX family

4.1.6 Debug outputs

Within the source, printf functions are called up for outputting the programme status information; outputs can be made via a serial interface using these functions. A corresponding function that initialises the interface is required for this.

4.2 Configuring the bootloader

The bootloader is configured via the BlCopCfg.h header file.

BLCOP_MAX_PROGRAMS:

This constant indicates the number of applications that can be programmed into the flash with the bootloader. The current version of the bootloader only supports one application.

BLCOP_MAX_PROGRAMS:

This constant indicates the number of bytes that can be transmitted to the bootloader in a block. The block information (block number, flash address, number of data bytes and block CRC) is included.

BLCOP_MIN_NODEID:

This constant indicates the smallest CANopen node address that is supported by the bootloader. The smallest possible value for this constant is 1 (limited by the CANopen Standard CiA 301).

BLCOP_MAX_NODEID:

This constant indicates the largest CANopen node address that is supported by the bootloader. The largest possible value for this constant is 127 (limited by the CANopen Standard CiA 301).

BLCOP_MIN_BAUIDX:

This constant indicates the smallest baud rate index that is supported by the bootloader.

BLCOP_MAX_BAUIDX:

This constant indicates the largest baud rate index that is supported by the bootloader.

BLCOP_SEND_BOOTUP:

This constant indicates whether or not the CANopen boot-up message is to be transmitted. In the release version of the bootloader (NDEBUG is defined), this constant is set to FALSE; the message is therefore not sent.

BLCOP_BDI_TABLE_PTR:

This constant indicates which baud rate table is to be used. The baud rate settings in the CAN controller vary when another CPU frequency is set. The baud rate settings must then be re-determined.

BLCOP_BDI_TABLE_SIZE:

This constant indicates the size of the baud rate table in bytes.

BLCOP_BASE_REQUEST:

This constant indicates the base CAN identifier for the SDO request that constitutes the actual CAN identifier together with the CANopen node address.

BLCOP_BASE_RESPONSE:

This constant indicates the base CAN identifier for the SDO response that constitutes the actual CAN identifier together with the CANopen node address.

BLCOP_USE_CANCRTL:

This constant indicates which CAN interface is to be used by the bootloader. The value 0 means CAN_A, value 1 means CAN_B and value 2 means CAN_C.

BLCOP_USE_CANINTENABLE:

This constant indicates which function must be called to temporarily block the CAN interrupt.

BLCOP_MAX_CANLOOPS:

This constant indicates the maximum number of CAN messages to be evaluated from the receive buffer. As deleting the flash sectors takes a relatively long time, this value should be set to 1.

BLCOP_IDENTITY_VENDORID:

This constant indicates the value for the Vendor ID to be written into the 0x1018/1 object. The value 0x3F (63 decimal) indicates the SYS TEC electronic GmbH company.

BLCOP_IDENTITY_PRODUCTID:

This constant indicates the value for the Product ID to be written into the 0x1018/2 object. The value 0x0610 (value specific to SYS TEC) indicates the CAN flash bootloader.

BLCOP_IDENTITY_REVISION:

This constant indicates the revision number to be written into the 0x1018/3 object. The value is to be indicated in the format as defined in the CANopen Standard CiA 301. For example, the value 0x00010002 would indicate Version V1.02.

BLCOP_IDENTITY_SERIALNR:

This constant indicates the serial number to be written into the 0x1018/4 object. According to the CANopen standard, each device must have a unique serial number. To be able to implement this, the serial number must be read out from an EEPROM set one time during production. For this method, the parameter dwSerialNr_p has been reserved for the BlCopInitialize() function.

BLCOP_MAX_CRC_STEP_SIZE:

This constant indicates the maximum number of bytes should be run through in the calculation of the CRC when running the BlCopProcess() function. The external watchdog must be triggered again after this call.

BLCOP_MAX_FLSWRITE_STEP_SIZE:

This constant indicates the maximum number of bytes should be run through ifor writing the data in the flash when running the BlCopProcess() function. The external watchdog must be triggered again after this call.

BLCOP_BOOTLOADER_START:

This constant indicates the start address in the flash at which the bootloader starts.

BLCOP_BOOTLOADER_END:

This constant indicates the last address in the flash at which there can be bootloader code To ensure that the bootloader can be programmed separately in the flash, this value must end at a flash sector.

BLCOP_BOOTLOADER_RAM:

This constant indicates the start address in RAM at which the interrupt vector table starts.

BLCOP_APPLICATION_START:

This constant indicates the start address in the flash at which the application starts. This address is the basis for calling the reset vector of the application

BLCOP_APPLICATION_END:

This constant indicates the last address in the flash at which the application can be programmed. This address must also include the application CRC and the size of the application. Both parameters are always in the last 8 bytes of the application area in the flash.

BLCOP_APPLICATION_RAM:

This constant indicates the start address in the RAM that is reserved for the application.

FLSDRV_START_APPLICATION:

Refer to constant BLCOP_APPLICATION_START. The flash driver receives its own constant for the start address of the application to ensure that it can also be used in other projects.

FLSDRV_END_APPLICATION:

Refer to constant BLCOP_APPLICATION_END. The flash driver receives its own constant for the end address of the application to ensure that it can also be used in other projects.

5 Flash tool

The host application is divided into the partial application BinaryBlock-Converter and BinaryBlock-Download.

5.1 BinaryBlock-Converter

This tool generates a block-oriented binary format from the output format specific to the toolchain (S record, hex file, elf file). The following parameters must be transmitted:

Call: BinaryBlockConv [options] in-file [out-file]

Options	Default	Meaning
-I <format_name>	ihex	This parameter defines the input file format.
-O <format_name>	binary	This parameter is reserved for future adjustments. The output format corresponds to the block binary format as described in this document.
--start_address <val>	0x000000	This value defines the start address of the address area to be considered during conversion. The value must be indicated in C notation.
--end_address <val>	0xFFFFFF	This value defines the end address of the address area to be considered during conversion. Values outside of the *start_address – end_address* area are not stored in the binary file. The value must be indicated in C notation.
--block_size <val>	0x000400	This value defines the number of bytes for each binary block. The value must be indicated in C notation.
--gap_fill <val>	val =0xFF	This parameter defines the value of the byte with which holes within a binary block are filled.
--adjust_start <val>	0x000000	The start address in the binary block can be changed by this value. The value must be indicated in C notation.
--help	-	Output of supported parameters and file formats

Construction of the binary file:

Block 0
Block 1
...
Block N
Block -1

Construction of block 0:

Offset	**Parameter**	**Content**
0x0000	Block Number	0
0x0004	Flash Address	0
0x0008	Data Size [bytes]	8
0x000C	Data bytes	reserved
0x0010		reserved
0x0014	Block CRC	CRC

Construction of block 1 .. N:

Offset	**Parameter**	**Content**
0x0000	Block Number	1 … N
0x0004	Flash Address	destination address
0x0008	Data Size [bytes]	n
0x000C	Data bytes	Data
…		
0x000C+n	Block CRC	CRC

Construction of block -1:

Offset	Parameter	Content
0x0000	Block Number	-1
0x0004	Flash Address	0
0x0008	Data Size [bytes]	8
...	reserved	32
0x0028	Data bytes	Size of application
0x002C		CRC application
0x0030	Block CRC	CRC

The CRC is calculated over the entire content of a block according to the polynomial 0xEDB88320. The start value is 0xFFFFFFFF. The same algorithm is used to calculate the CRC for the entire application.

If an application is comprised of several partial applications then these must be generated from one <in-file> or several in-files by entering the address area. Each partial application must then be assigned a programme number that corresponds to the sub-index within the object's programme data.

5.2 BinaryBlock-Download

This tool transfers the binary blocks from the output file generated by the BinaryBlockConv. This tool thereby transfers exactly one partial application that is designated by its programme number. A USB-CAN module is used as the interface to the CAN-bus.

Call: BinaryBlockDownload [options] in-file

Options	Default	Meaning
-P < val >	1	This parameter defines the programme number. It corresponds to the sub-index for transmission of the command and programme data.
-B <index>	4	This parameter defines the bitrate to be used via the index. (4=125kBit/s, 3=250kBit/s)
-N < val >	1	This value defines the node address of the selected CANopen device.
--delay < val >	1000	This value defines the delay for requesting the status information after deleting or programming the flash.
--repeats < val >	0x000400	This value in ms defines the number of repetitions for requesting the status information after deleting or programming the flash.
--ser_num <val>	0	Serial Number for configuration of node numbers using LSS
--rev <val>	0	Revision code for configuration of node numbers using LSS
--pcode <val>	0	Revision code for configuration of node numbers using LSS
--vendor <val>	0x3F	Vendor-ID for configuration of node numbers using LSS
--timeout <val>	500	Max. timeout in ms for waiting from response from the target
--help	-	Output of supported parameters and file formats

There is a command shell integrated into the BinaryBlockDownload tool that allows the user to run various commands for controlling the download.

The following steps must be executed for download:

1. Start the bootloader during download

Starting the bootloader (re-entry into the bootloader on the target) is repeated until a bitrate is configured and the device type of the bootloader could be read. As the bitrate is automatically recognised it may be that the target is not able to receive commands. This is only possible after the bitrate can be recognised.
Furthermore, after the first time the bootloader is started, the target must be assigned a CANopen node ID. This can be done via the LSS service.
It may also be that an application is active.
An attempt is made to start the downloader when starting the download. This procedure is repeated until the bootloader is recognised or until it is cancelled by the user.

2. Bootloader has been started

Delete programme and wait until it is deleted (status prompt)

3. Transmit block-for-block from the BIN file and wait until the block is programmed (status prompt)
4. After the last block, the CRC is calculated on the target and stored in the OD. The CRC is read-out and compared to the value on the PC.
5. If no error has previously occurred then the application can be test run.
6. Wait a little until the application starts. Then check whether the application has been started (read the device type and compare, read the ident-object and compare).
7. Then switch to the bootloader and if no error has as yet been determined (device type OK, CRC OK, Ident-Object OK, test run OK) then set the signature and wait until it is programmed (status prompt).
8. If there is no error then the application can be started.

5.3 Configuring the flash tool software (FtCfg.h)

FT_FLASH_START:

This constant has the same meaning for the flash tool as constant BLCOP_APPLICATION_START for the bootloader.

FT_FLASH_SIZE:

This constant indicates the number of bytes in the flash area of the application.

FT_FLASH_DEF_VAL:

This constant indicates the value received in the flash after deletion. This value is important in the flash tool to calculate the CRC, whereby non-programmed flash cells must also be considered.

FT_BLOCK_SIZE:

This constant has the same meaning for the flash tool as constant BLCOP_MAX_PROGRAM_BUFFER for the bootloader.

FT_BLOCK_MAX_GAP:

This constant sets the maximum number of non-programmed bytes before the flash tool uses a new block for downloading the subsequent programme data. It is indented to prevent 16384 bytes having to be transmitted in one block to the bootloader although, e.g. only the fist and the last byte have to be programmed according to the HEX file.

FT_DEF_NODE_ID:

If the call parameter –N... is not transmitted to the flash tool, then the HEX file is transmitted to the CANopen node address defined in this constant. Constants FT_BASE_CANID_SDO_REQUEST ad FT_BASE_CANID_SDO_RESPONSE are included for the CAN identifier used.

FT_DEF_NODE_ID:

If call parameter –B... is not transmitted to the flash tool then the baud rate index of this constant is used.

FT_DEF_RETRY_TIMEOUT:

This constant indicates the number of repetitions that are carried out during SDO transfer after a timeout.

FT_DEF_RETRY_BUSY:

This constant indicates the number of repetitions carried out by SDO transfer after the bootloader has returned the BUSY state.

FT_DEF_TIME_DELAY:

This constant indicates the time delay in 100 milliseconds steps that the flash tool waits between reading object 0x1F57/1. This is intended to prevent that the flash tool does not fill the CAN bus unnecessarily with CAN messages while the bootloader is busy.

FT_DEF_SDOC_TIMEOUT:

This constant indicates the time delay in 100 milliseconds steps until the flash tool determines a timeout during SDO transfer.

FT_BASE_CANID_SDO_REQUEST:

This constant has the same meaning for the flash tool as constant BLCOP_BASE_REQUEST for the bootloader.

FT_BASE_CANID_SDO_RESPONSE:

This constant has the same meaning for the flash tool as constant BLCOP_BASE_RESPONSE for the bootloader.

FT_MIN_NODE_ID:

This constant has the same meaning for the flash tool as constant BLCOP_MIN_NODEID for the bootloader.

FT_MAX_NODE_ID:

This constant has the same meaning for the flash tool as constant BLCOP_MAX_NODEID for the bootloader.

5.4 Error codes of the flash tool software (FtErrDef.h)

The error codes of functions in the flash tool and in flashtool.exe itself (ERRORLEVEL system variable) are compatible to the POSIX standard. If bit 30 is set in the error code then this is a user-specific error code. Otherwise it is an error code from the operating system.

Error codes from the flash tool software begin from value 0x60000000. All possible error codes are listed in table X.

However, if for instance the input file could not be opened then the flash tool reads the error code from the „errno“ variable and returns it to the console together with an appropriate error message. A list of possible error codes is contained in the MSDN.

Error code	Name: Meaning
0x00000000	No error
0x60000001	ERR_BL_NO_VALID_PROGRAM: The bootloader reported an invalid application. This happens when the bootloader calculates another application CRC than the flash tool calculates.
0x60000002	ERR_BL_DATA_FORMAT_UNKNOWN: The bootloader reported an unknown data format in block acc. to ***Figure 2***.
0x60000003	ERR_BL_CRC_ERROR: The bootloader reported a CRC error in the data block.
0x60000004	ERR_BL_FLASH_NOT_CLEARED: The bootloader reported that the flash area to be programmed is not cleared. The blank check failed or the command to delete the application has not been transmitted.
0x60000005	ERR_BL_FLASH_ERROR: The bootloader has determined a general error (not specified in more detail) when writing the data into the flash.
0x60000006	ERR_BL_ADDRESS_ERROR: The bootloader has determined that an attempt is made to write to the area above the flash area of the application.
0x60000007	ERR_BL_FLASH_SECURED: The bootloader has determined that an attempt has been made to write to the protected flash area reserved for the bootloader itself.

Error code	Name: Meaning
0x60000008	ERR_BL_ERROR: The bootloader has determined an error unknown to the flash tool.
0x60000010	ERR_NO_BOOTLOADER_RUNNING: The flash tool can not address the bootloader. The bootloader is possibly inactive.
0x60000011	ERR_USER_ABORT: The user has aborted the transfer.
0x60000012	ERR_DOWNLOAD_RUNNING: Function FtCopStartDownload() has been called although there is still an active download to the bootloader.
0x60000013	ERR_NODE_NOT_FOUND: An SDO timeout has been determined. The bootloader is not responding to a request.
0x60000014	ERR_START_FAILED: The start command could not be transferred to the bootloader.
0x60000015	ERR_PROGRAM_TOO_LARGE: The flash tool has determined that an attempt is made to write to the area above the flash area of the application. The addresses in the Motorola HEX file are thereby checked.
0x60000016	ERR_FLASHING_WO_END: The flash tool has read the flash status from the bootloader a multiple number of times (*refer to object 0x1F57/1 in **Table 1***). The number of attempts has expired, but the bootloader still reports a BUSY state. Either the value of the constant FT_DEF_RETRY_BUSY must be increased or the bootloader is hung.
0x60000017	ERR_INVALID_NODEID: An API function from the FlCop.c module has been called with an invalid CANopen node address.
0x60000018	ERR_INVALID_PARAM: An API function from the FlCop.c module has been called with an invalid parameter (e.g. ZERO pointer).
0x60000020	ERR_SREC_CRC_ERROR: The flash tool has determined a checksum error in the Motorola HEX file.
0x60000021	ERR_SREC_NO_VALID_S3: The flash tool has determined that the indicated file does not have the S3 format.
0x60000022	ERR_SREC_FORMAT_ERROR: The flash tool has determined an error in the S3 format.

Error code	Name: Meaning
0x60000023	ERR_SREC_OVERLAP: The flash tool has determined that at least two areas overlap in the Motorola HEX file.
0x60000024	ERR_SREC_ADDESS_RANGE: The HEX file indicated has the S3 format, but there is no entry in this HEX file that fits into the flash area of the application.
0x60000025	ERR_SDO_ERROR Error during SDPO access/transfer.
0x60002001	WARN_SREC_ADDRESS_RANGE: This warning is returned by the FtCopStartDownload() or FtBinOpenFile() function if some entries in the Motorola HEX file are not written into the flash area of the application. These entries are ignored, i.e. not transferred to the bootloader. The application may not be able to be run.

Table 5: Meaning of the flash tool error codes

All error codes from 0x60001000 flag an error code from the CANopen stack. The meaning of the corresponding error code (minus offset 0x60001000) can be looked up in L-1020.

6 Resources

6.1 Code, data target

The actual values depend on the compiler, the memory model supported, the optimisation level and the CPU. To run the bootloader, there should be 32kByte flash and approx. 6kByte RAM available on a 16bit system.

If the existing resources are insufficient in order to integrate the bootloader then the software modules can be adjusted within the framework of an adaptation workshop.

6.2 Target interrupts

The bootloader is based on the standard CAN driver for CANopen. Receiving and sending messages as well as changing the CAN controller status, is signalised by interrupts. If, due to additional tasks of the microcontroller during the update procedure, the use of interrupts not be supported, then the appropriate CAN driver routines must be adjusted by the user. An adjustment can be made within the framework of the adaptation workshop in cooperation with SYS TEC.
Furthermore, a system time is required; this is generally controlled by interrupts.

As the bootloader and the applications transferred by the bootloader share the interrupt for the CAN controller, appropriate adjustments must be made in the form of interrupt forwarding or interrupt vector tables in the RAM. depending o the CPU used, the interrupt vector table may have to be mirrored. Another variation of this would be to construct this table in RAM and every driver then enters the vector of the interrupt service routine there. This allows the bootloader to modify the vectors when using interrupts for the timer and the CAN controller. On starting the application, the relevant entries are then replaced by the application vectors. When preparing to integrate the bootloader, the respective procedure must be agreed upon and

an example programme is to be implemented by the project partner that demonstrates this implementation.

Document:	CANopen Bootloader
Document number:	**L-1112e_5**, Edition August 2008

How would you improve this manual?

Have you found mistakes in this manual? Page

Sent by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Send to:

SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
GERMANY
Fax: +49 (0) 36 61 / 62 79 99

Published by

SYS TEC ELECTRONIC

Order No. L-1112e_5
Printed in Germany