
TM

# *EtherCAT Realtime Master Library Documentation (Cluster 32/64 Bit)*

Date: Oct, 29.2014

# EtherCAT Realtime Master Library Documentation

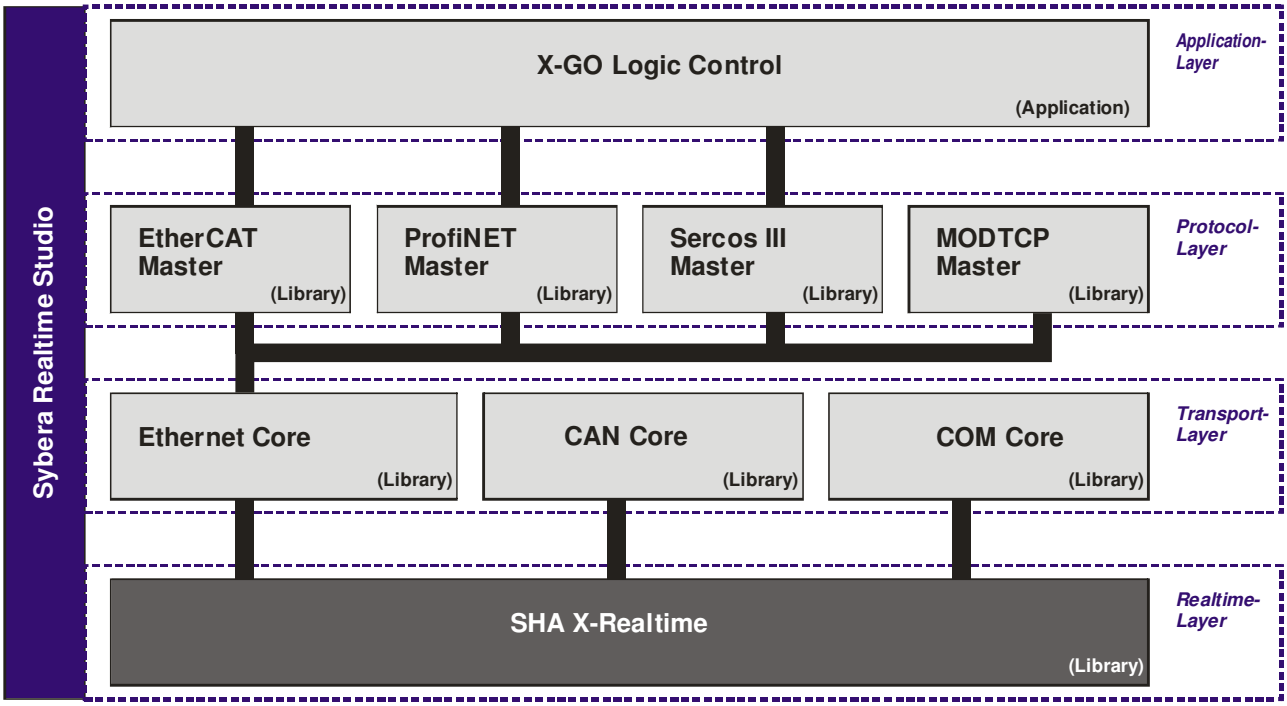
SYBERA Copyright © 2014

SYBERA Copyright © 2014

SYBERA Copyright © 2014

# 1 Introduction

The idea of further interface abstruction of the SHA X-Realtime for several communication channels and bus systems, like serial communication, CANBUS, Ethernet (TCP/IP), … is realized by the SYBERA AddOn Software Moduls, so called RealtimeCores. All RealtimeCores are based on the SHA X-Realtime system. The RealtimeCores are intended to fullfill Realtime-Level-1, which means collecting and buffering data in realtime without loss of data, as well as Realtime-Level-2, which means functional operation at realtime. Thus the RealtimeCores usually require simple passive harware. One of the great benefits is the adjustable scheduling time of incoming and outgoing data.



SYBERA Copyright © 2014

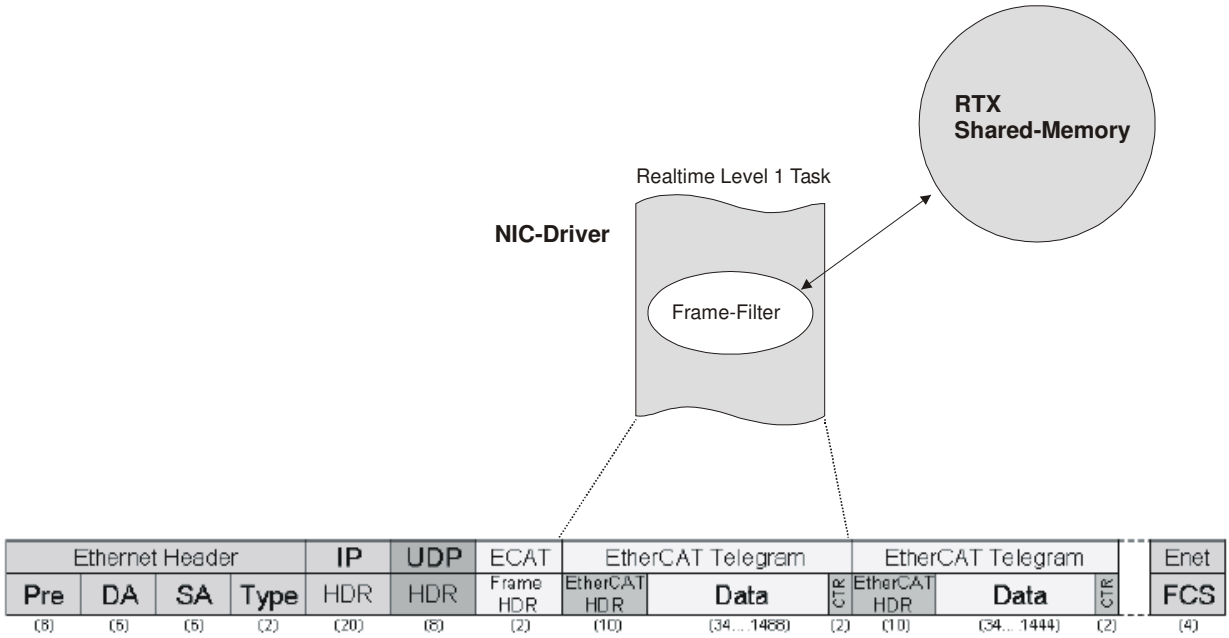
The EtherCAT realtime library system of SYBERA enables a custom ethernet adapter under Windows as an EtherCAT Master. Therefore the base is the Sybera X-Realtime technology. The library system allows the deterministic control of EtherCAT slave participants (e.g., the EtherCAT modules from Beckhoff Automation GmbH). Depending on the PC hardware telegram update cycles upto 50 microseconds are realistic. As physical link customary INTEL or REALTEK chips are suitable.

Beside numerous extended EtherCAT functions for Distributed Clock, COE and State management, the library system also allows to control EtherCAT devices, even without a corresponding XML file. With the integrated station management the devices may be completely administered and controlled almost implicitly, or every single functional step (FMMU, SYNCMAN, PDO, STATE...) may be controlled separately. In addition, SYBERA has developed the comprehensive test software ECATVERIFY which allows the developer to test Ethercat devices without programming and to parametrize the devices. Thereby the developer is led through the startup procedure interactively by single functional groups and states. Besides, all information are visualised in detail.

On this occasion, not only the sending and receiving of ethernet frames under realtime condition due to the specification of the EtherCAT Technology Group (ETG) is realized. The interface allows the functional control of EtherCAT telegrams in a separate realtime task. The system is based on 4 realtime tasks, for sending and receiving of ethernet frames, error management and functional control. With an integrated state machine the tasks are functionally synchronized. A realtime error task recognizes any frame failure and hardware latency. It is checked if an answer was received to a sent telegram (integrated timeout condition), if the working counter of the answer telegram is 0 and if the index fields of the sending telegram match the and receiving telegram. In addition, an emergency telegram is deposited, being sent by the error task in case of an error condition. A frame filter will separate the EtherCAT telegrams within the ethernet frame and transfer them to the telegram stack.

SYBERA Copyright © 2014

One or more EtherCAT telegrams are embedded in an ethernet frame. On sending the Realtime Core pops the EtherCAT telegrams from the EtherCAT interface stack and build them inside an ethenet frame. On receiving the EtherCAT telegrams will be extracted from the ethernet frame and pushed to the EtherCAT stack.



SYBERA Copyright © 2014

## 1.1 Product Features

- Intelligent Station Management
- Station Realtime Cycles upto 50 µsec
- Logical, Physical and Alias Station Addressing
- Mailbox Interface and COE Management
- PDO Assignment
- Integrated PDI Control
- FMMU Management
- SYNC Management
- Distributed Clock Support
- Watchdog Support
- State Management
- XML, SII and Native Station Configuration
- HighLevel EtherCAT Interface
- LowLevel EtherCAT Interface

## 1.2 Supported Platforms

SHA was build to support serveral development platforms. Currently following platforms are supported:

- Visual C++ (from Version 2008)
- CVI LabWindows
- Borland C++Builder

## 1.3 Supported OS

-
- Windows XP, VISTA, 7, 8 (32 / 64 Bit)

SYBERA Copyright © 2014

# 2 EtherCAT Library Installation

For installation following steps are required:

### Preparation

1. Provide a PC with INTEL or REALTEK Ethernet adapter and Windows operating system (with administrator privileges)

### Installation

2. Install SHA realtime system (separate software package)

3. Install ETH transport library (separate software package)

4. Run the program SYSETUP(32/64) of the master library
   (make sure the directory path has no space characters)

   On Installation the PEC information (PID, SERNUM and KEYCODE) must be entered. The KEYCODE for the evaluation version is: 00001111-22223333

5. Optional: Check license with SYLICENCECHECK(32/64).EXE

### Operation

6. Run ECATVERIFY(32/64).EXE (with admistrator privileges)

7. Build device description ECATDEVICE.PAR
   (must be placed in C:\WINDOWS\SYSTEM32)

8. Build the program with the library interface

9. Run the program

*Note:* After finishing installation, you must reboot your PC before starting the compiler !!!.

Note: In order to operate SYBERA software under Windows 8, 7, VISTA, it must be carried out with ADMINISTRATOR priviledges.

SYBERA Copyright © 2014

Note: For proper operation, make shure within the BIOS the *INTEL Speedstep Technologie*, the *INTEL TurboBoost Technologie* as well as the *INTEL C-STATE Technologie* is turned off.

*Enhanced SpeedStep — SpeedStep also modulates the CPU clock speed and voltage according to load, but it is invoked via another mechanism. The operating system must be aware of SpeedStep, as must the system BIOS, and then the OS can request frequency changes via ACPI. SpeedStep is more granular than C1E halt, because it offers multiple rungs up and down the ladder between the maximum and minimum CPU multiplier and voltage levels.*

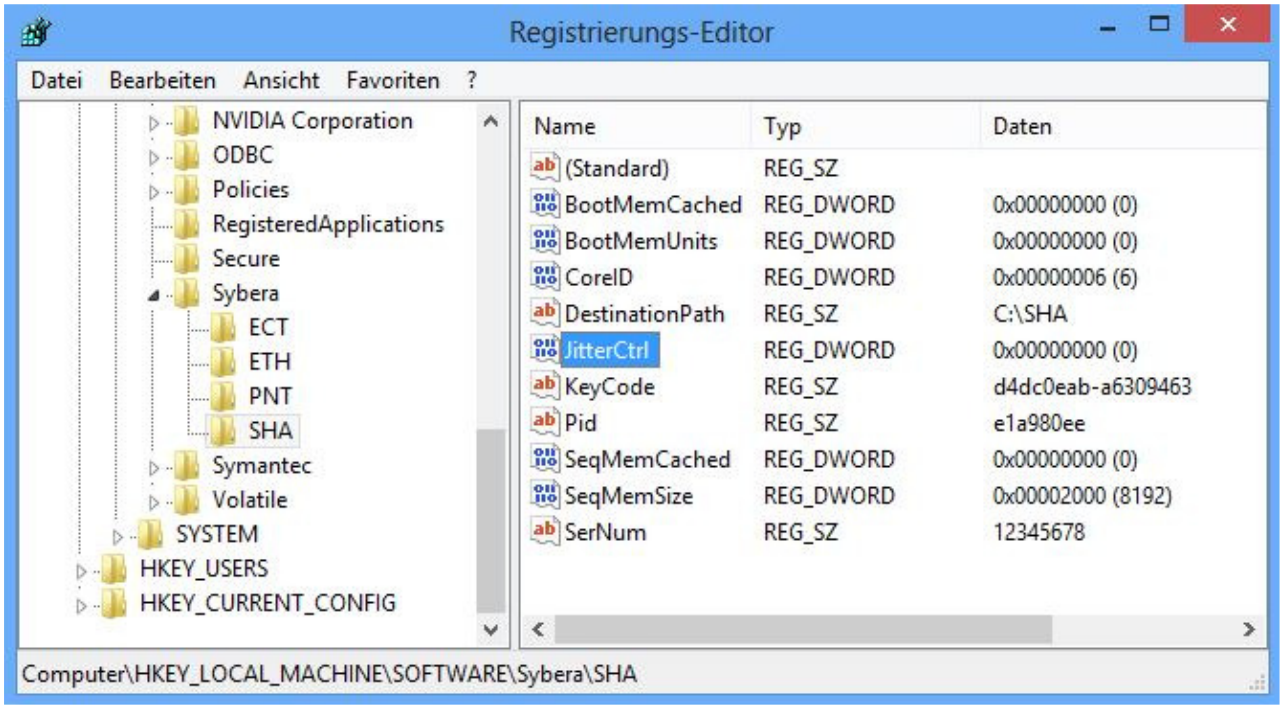
*C1E enhanced halt state — Introduced in the Pentium 4 500J-series processors, the C1E halt state replaces the old C1 halt state used on the Pentium 4 and most other x86 CPUs. The C1 halt state is invoked when the operating system's idle process issues a HLT command. (Windows does this constantly when not under a full load.). C0 is the operating state. C1 (often known as Halt) is a state where the processor is not executing instructions, but can return to an executing state essentially instantaneously. All ACPI-conformant processors must support this power state. Some processors, such as the Pentium 4, also support an Enhanced C1 state (C1E or Enhanced Halt State) for lower power consumption. C2 (often known as Stop-Clock) is a state where the processor maintains all software-visible state, but may take longer to wake up. This processor state is optional. C3 (often known as Sleep) is a state where the processor does not need to keep its cache coherent, but maintains other state. Some processors have variations on the C3 state (Deep Sleep, Deeper Sleep, etc.) that differ in how long it takes to wake the processor. This processor state is optional.*

*Intel® Turbo Boost Technology automatically allows processor cores to run faster than the base operating frequency, increasing performance. Under some configurations and workloads, Intel® Turbo Boost technology enables higher performance through the availability of increased core frequency. Intel® Turbo Boost technology automaticallyallows processor cores to run faster than the base operating frequency if the processor is operating below rated power, temperature, and current specification limits. Intel® Turbo Boost technology can be engaged with any number of cores or logical processors enabled and active. This results in increased performance of both multi-threaded and single-threaded workloads.*

SYBERA Copyright © 2014

## 2.1 Jitter Control

Since a notebook has a quiet different jitter behaviour than desktop systems, an enhanced jitter control mechanism is required. Therefore SYBERA provides a registry entry called “JitterCtrl”. This entry allows an adaptive iteration to the best jitter behaviour of the notebook.



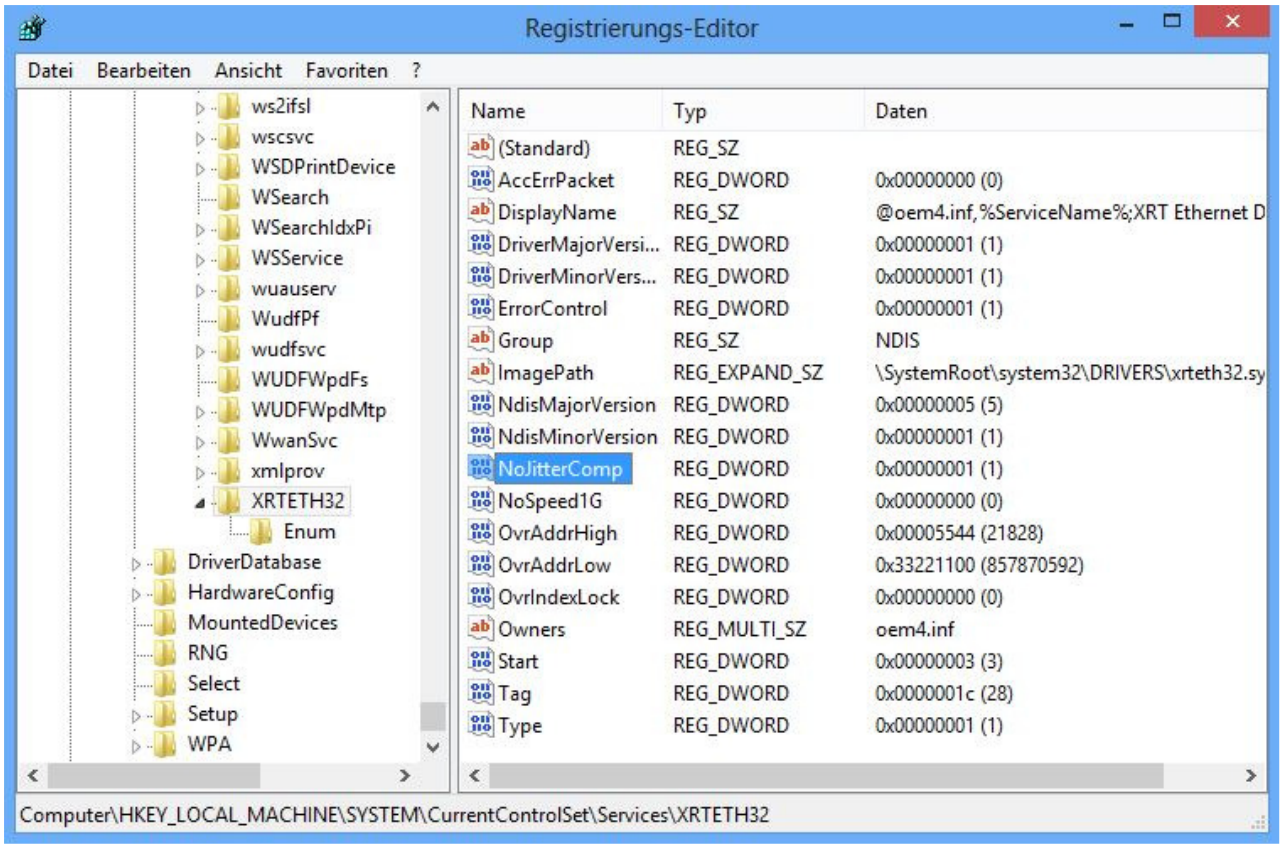

Following values are valid:

0: No enhanced jitter control
1: Enhaced Jitter Control, Step 1 (first choice together with BIOS settings)
2: Enhaced Jitter Control, Step 2 (for INTEL platforms only)
3: Enhaced Jitter Control, Step 3 (for INTEL platforms only, together with BIOS settings)

SYBERA Copyright © 2014

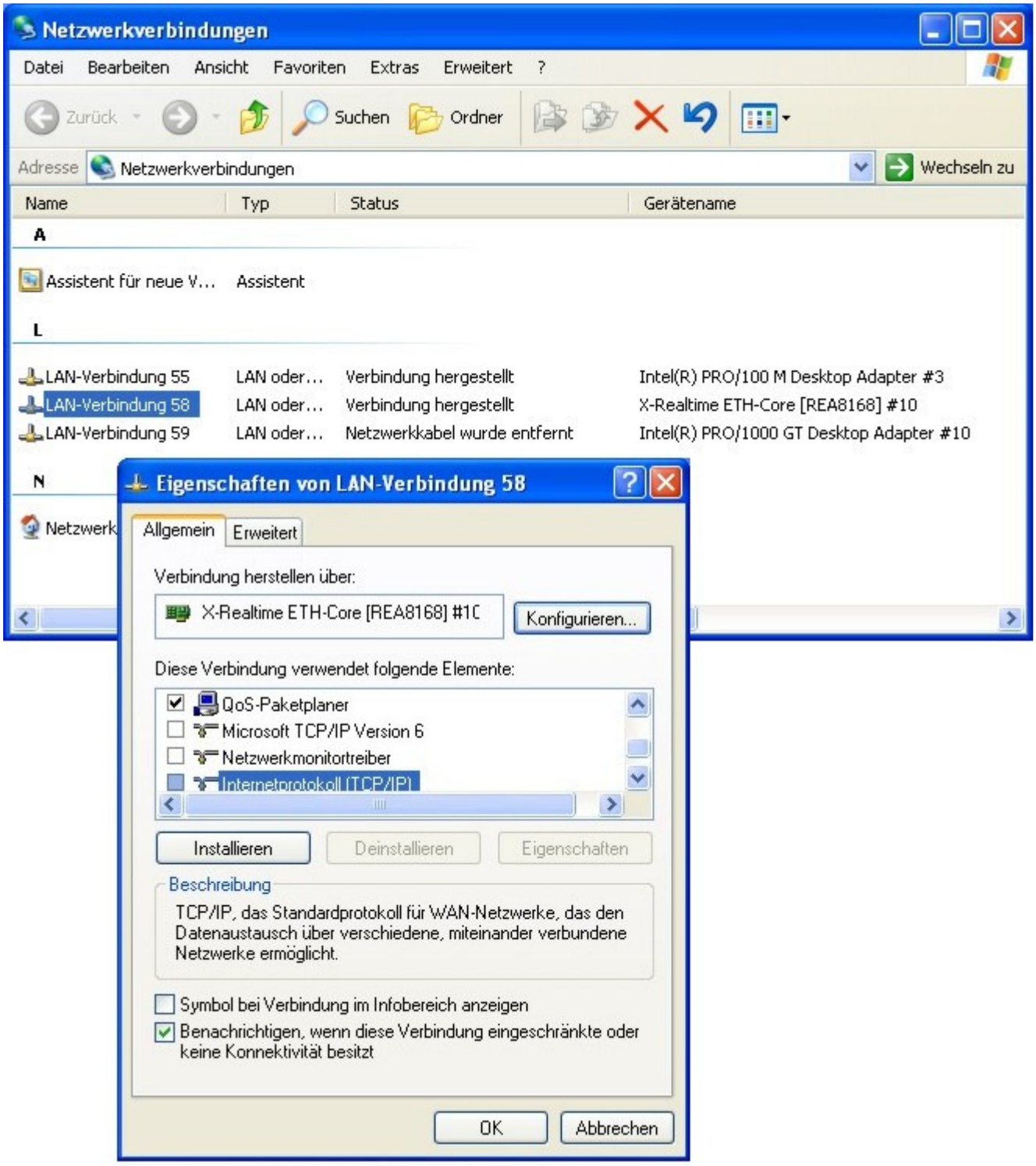
## 2.2 Dynamic Jitter Compensation

SYBERA uses the procedure "Dynamic Jitter Compensation" with active and passive feedback compensation within the realtime engine. Although the X-Real time engine of SYBERA allows a native maximum Jitter of approx. 15 µ sec (according to hardware platform), this behaviour may be reduced below 3 µsec by the dynamic jitter compensation.
For compatibility reason on some platforms it may be required to disable the dynamic jitter compensation. Therefore the registry value “NoJitterComp” has to be set to 1



SYBERA Copyright © 2014

*Note:* For proper operation its recommended to use the EtherCAT network as standalone network. This requires to turn off the Windows protocols for this network connection:



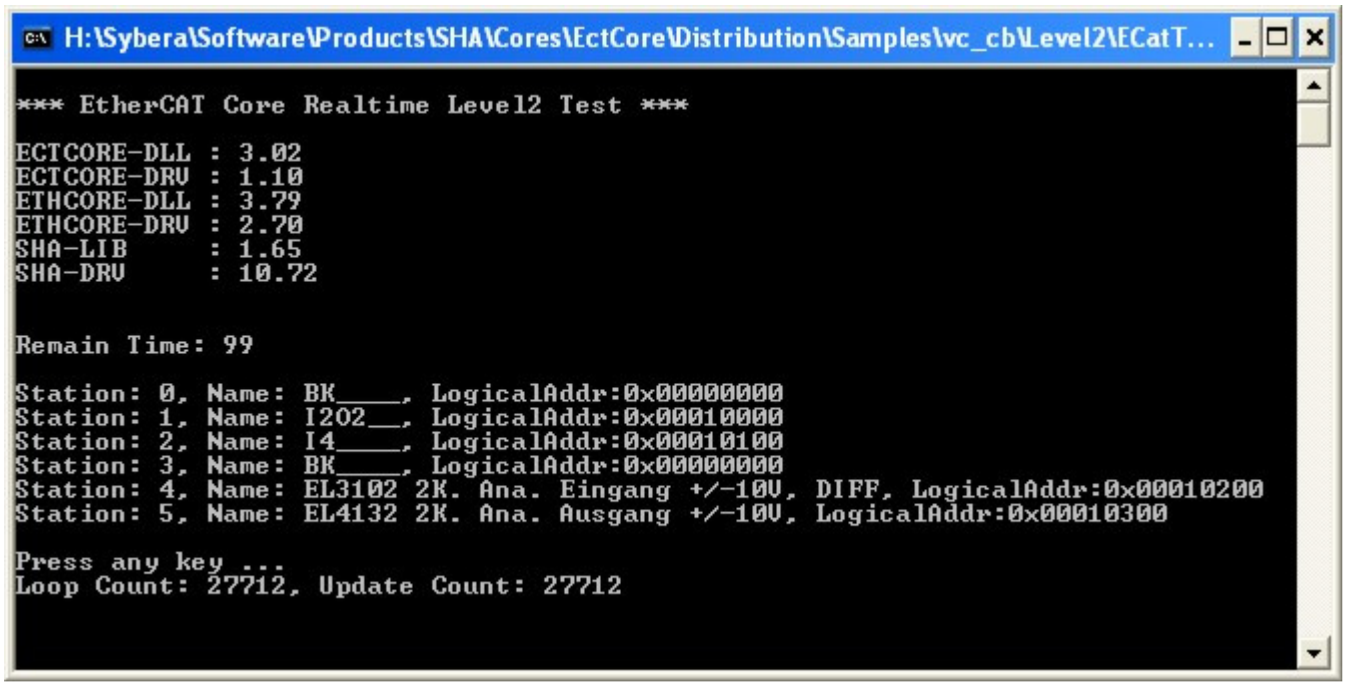
SYBERA Copyright © 2014

# 3 EtherCAT Realtime Master Library

The interface functions of the EtherCAT Realtime Master Library are exported by a static link library. Following include files and libraries are required:

| | |
|---|---|
| SHA*(32/64)*ECATCORE.DLL | EtherCAT Master DLL (VISUAL C++) |
| SHA*(32/64)*ECATCORE.LIB | EtherCAT Master LIB (VISUAL C++) |
| SHA*(32/64)*ECATCOREOML.DLL | EtherCAT Master DLL (BORLAND C++ / Delphi) |
| SHA*(32/64)*ECATCOREOML.LIB | EtherCAT Master LIB (BORLAND C++ / Delphi) |
| ECAT*(32/64)*DEVICE.PAR | Native Station Configuration File |
| SHA*(32/64)*ECATCORE.H | Exported Function Prototypes |
| ECAT*(32/64)*COREDEF.H | EtherCAT Basic Definitions |
| ECAT*(32/64)*SDODEF.H | EtherCAT COE Definitions |
| ECAT*(32/64)*SIIDEF.H | EtherCAT EEPROM Definitions |
| ECAT*(32/64)*DCDEF.H | EtherCAT Distributed Clock Definitions |
| ECAT*(32/64)*REGDEF.H | EtherCAT Register Definitions |
| ECAT*(32/64)*MAILBOXDEF.H | EtherCAT Mailbox Definitions |
| ECAT*(32/64)*MACROS.H | EtherCAT Macro Definitions |
| ECATDBG.LOG | Sequence Log (generated at runtime) |

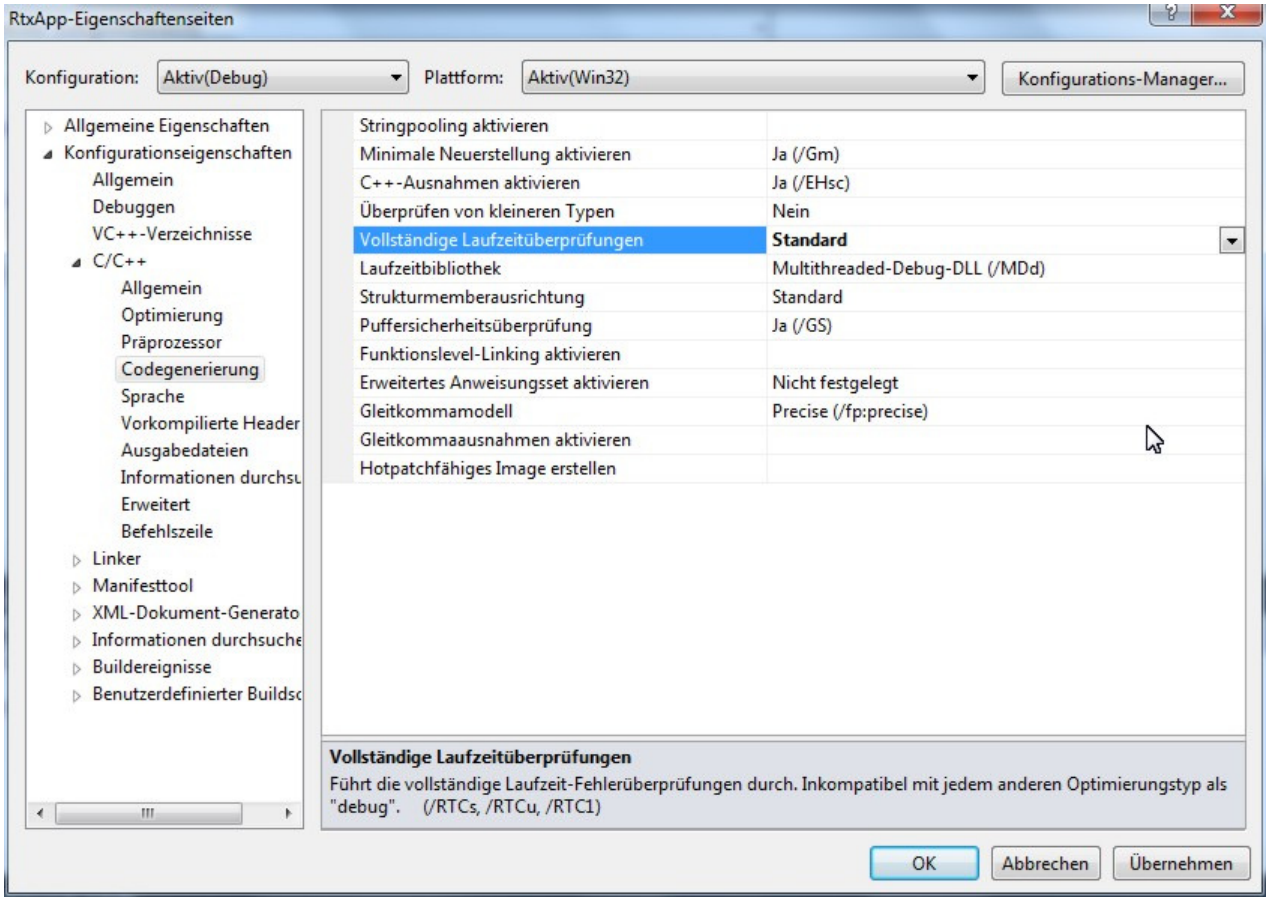
Sample Application



SYBERA Copyright © 2014

### 3.1.1 Visual Studio 2010 Compiler Settings

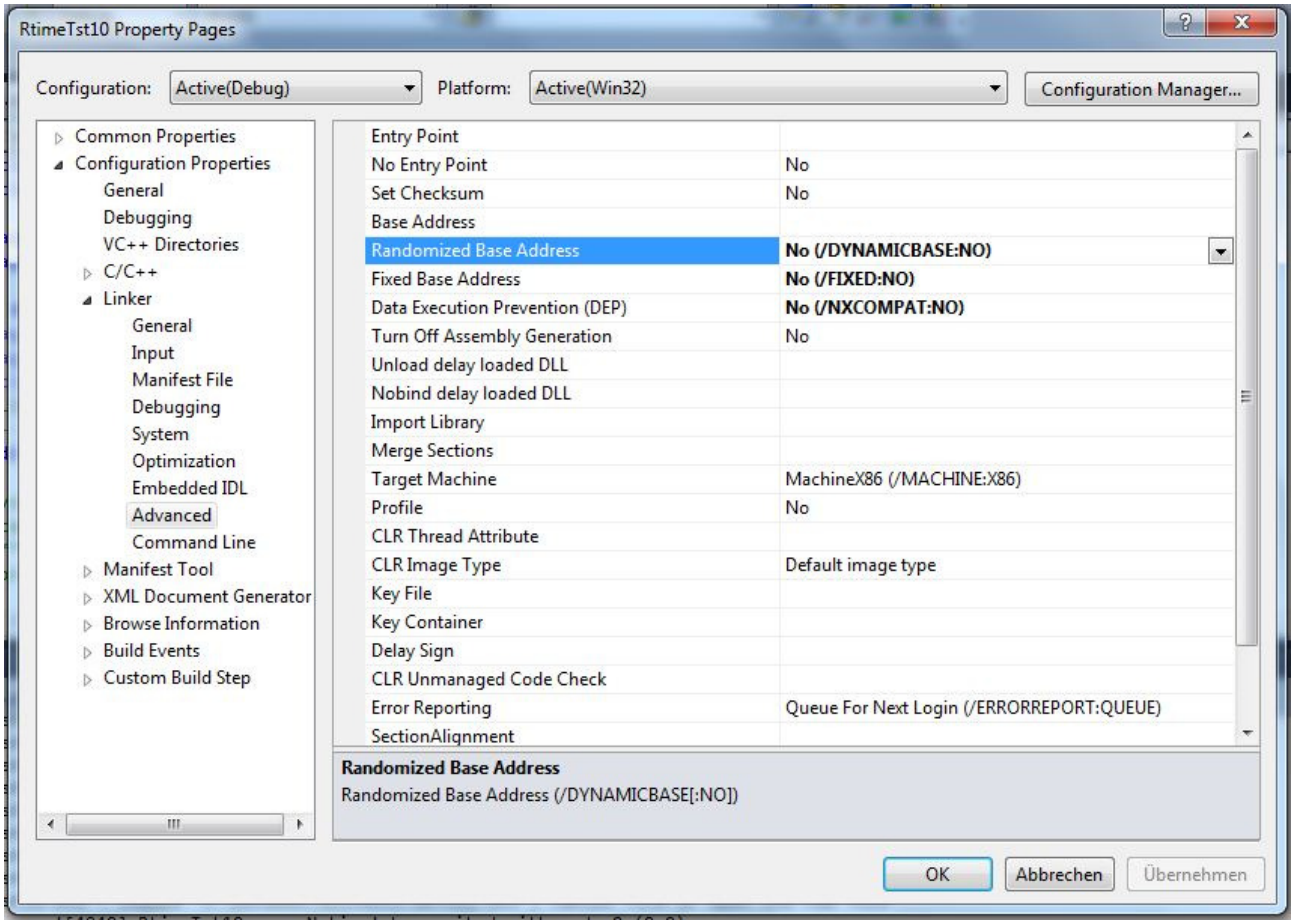
With Visual Studio 2010 a change in the COMPILER settings was introduced. To make the Virtual Code Mapping (VCM) working correctly, the settings must be changed:



SYBERA Copyright © 2014

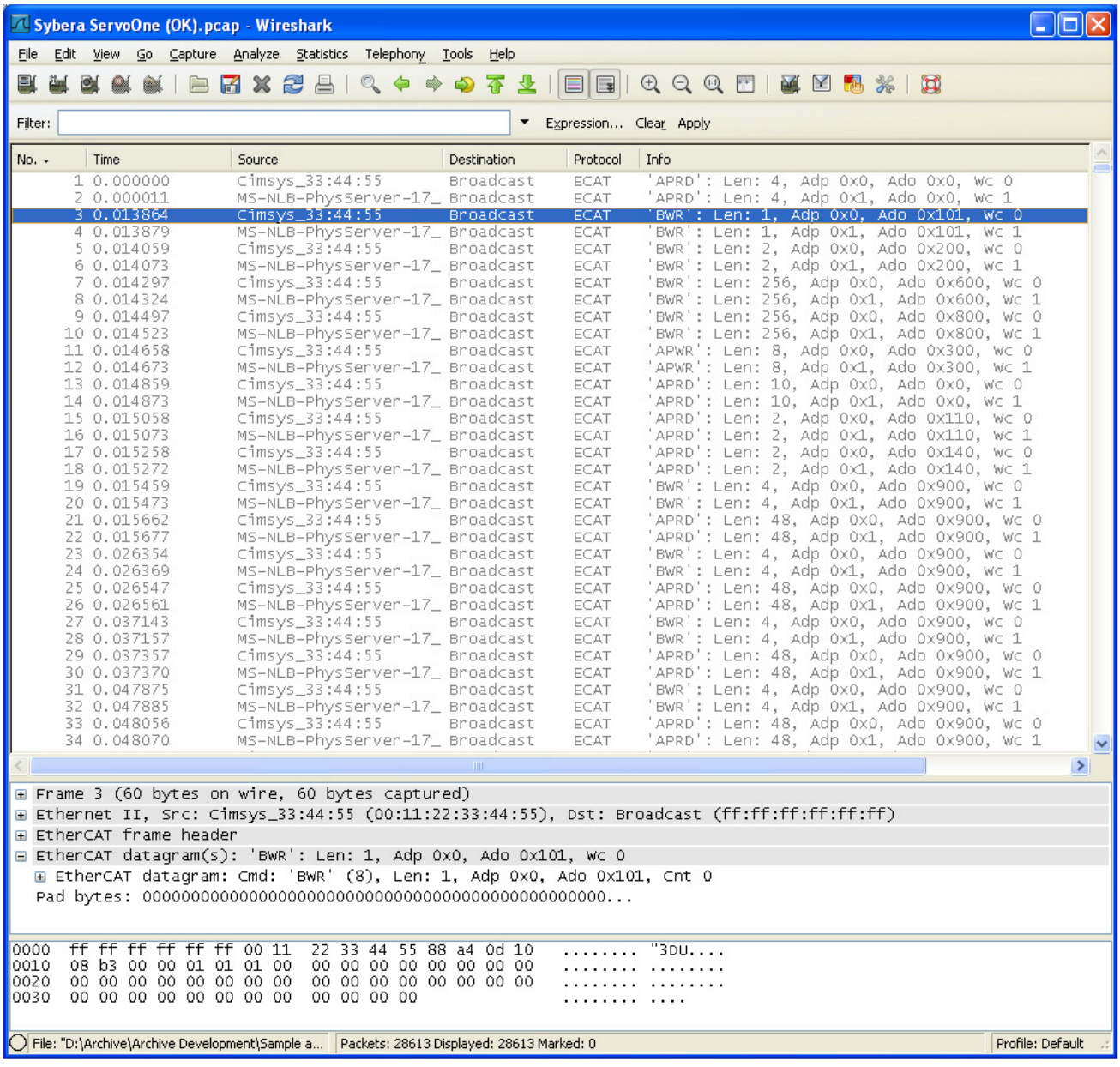
### 3.1.2 Visual Studio 2010 Linker Settings

With Visual Studio 2010 a change in the LINKER settings was introduced. To make the Virtual Code Mapping (VCM) working correctly, the settings must be changed:



SYBERA Copyright © 2014

Sample Startup Protocoll:



SYBERA Copyright © 2014

## 3.2 Header File ECAT(32/64)COREDEF.H

The header file ECAT(32/64)COREDEF.H is required when handling EtherCAT telegrams by the interface functions or handling the EthernetCore Realtime Stack directly (Realtime Level2). It also defines the EtherCAT telegram commands and structures.

### 3.2.1 Structure ECAT_PARAMS

This structure is required by the HighLevel Interface functions, and contains all required and optional input and output data members.

```
typedef struct _ECAT_PARAMS
{
      //Input parameters
      USHORT            FixedAddr;        //Fixed Station Address
      ULONG             LogicalAddr;      //Logical Station Address
      ULONG             SyncCycles;       //Cycles for synchronisation interval

      //Output parameters
      ULONG             ErrCnts;          //Error Counters
      FP_ECAT_ENTER     fpEcatEnter;      //Function Pointer to EcatEnter()
      FP_ECAT_EXIT      fpEcatExit;       //Function Pointer to EcatExit()
      ULONG             core_dll_ver;     //Core DLL version
      ULONG             core_drv_ver;     //Core driver version

      //Input - Output parameters
      ETH_PARAMS        EthParams;        //Ethernet Core Parameters

      //Realtime level2 parameters
      SHORT             StationNum;       //Station Number
      PSTATION_INFO     pSystemList;      //Station List Pointer
                                          //(use inside Realtime Task)
      PSTATION_INFO     pUserList;        //Station List Pointer
                                          //(use outside Realtime Task)
} ECAT_PARAMS, *PECAT_PARAMS;
```

Note:

The structure ETH_PARAMS is part of the Ethernet Core Library and described in the the documentation of this core library. Thus the Ethernet Core library must be installed first. The required elements of the structure ETH_PARAMS must be used in the same way as using the elements of ECAT_PARAMS.

SYBERA Copyright © 2014

### 3.2.2 Structure STATION_INFO

This structure keeps all information of each EtherCAT modul and may be required for further interface functions.

```
typedef struct _STATION_INFO
{
     char             szName[MAX_PATH_SIZE];         //Name of Station
     USHORT           Index;                         //Station Index
     AL_CONTROL       AlControl;                     //AL Control
     AL_STATUS        AlStatus;                      //AL Status
     DL_CONTROL       DlControl;                     //DL Control
     DL_STATUS        DlStatus;                      //DL Status
     DL_INFORMATION   DlInfo;                        //DL Information
     PDI_CONTROL      PdiControl;                    //PDI Control
     PDI_CONFIG       PdiConfig;                     //PDI Configuration
     SII_AREA_HDR     SiiAreaHdr;                    //SII Area Information
                                                     //(Header)
     DC_LOCAL_TIME    DcLocalTime;                   //DC Local Time
     DC_SYNC_INFO     DcSyncInfo;                    //DC Sync Information
     RX_ERR_CNT       RxErrCnt;                      //RX Error Counter
     FMMU             FmmuList[MAX_FMMU_NUM];        //FMMU Manager List
     ULONG            FmmuNum;                       //Number of FMMU records
     SYNCMAN          SyncManList[MAX_SYNCMAN_NUM];  //SYNCMAN Manager List
     ULONG            SyncManNum;                    //Number of SYNCMAN records
     SDO_LEGACY       SdoList[MAX_SDO_NUM];          //SDO Legacy Command List
     ULONG            SdoNum;                        //Number of SDO commands
     USHORT           PhysAddr;                      //Physical Station Address
     USHORT           AliasAddr;                     //Alias Station Address
     ECAT_TELEGRAM    TxTel;                         //TX Process Telegram
     ECAT_TELEGRAM    RxTel;                         //RX Process Telegram
     DATA_DESC        OutDescList[MAX_DATA_DESC];    //Output Descriptor List
     ULONG            OutDescNum;                    //Number of TX Data
                                                     //Descriptors
     DATA_DESC        InDescList[MAX_DATA_DESC];     //Input Descriptor List
     ULONG            InDescNum;                     //Number of RX Data
                                                     //Descriptors
     BOOLEAN          bUpdate;                       //Station Update Flag
                                                     //(read only)
     BOOLEAN          bDisable;                      //Station Disable Flag
     UCHAR            Reserved[MAX_RESERVED_SIZE];   //Reserved Data Size

} STATION_INFO, *PSTATION_INFO;
```

SYBERA Copyright © 2014

Note:

- The EtherCAT structures (AL_CONTROL, AL_STATUS, DL_STATUS, …) are described in detail inside the EtherCAT specification and are only used for the development with the EtherCAT Library LowLevel Interface.

- Since most Library LowLevel Routines effect all stations, each station may be disabled by setting the flag *pStation->bDisable = TRUE* to be unaffected by the functions

- The flag *pStation->bUpdate* is used to check if the station has been updated, especially when more Ethernet frames are required for updating all stations

- The field reserved may be used for station specific data and has the size of MAX_RESERVED_SIZE

- For accessing the realtime process telegrams TxTel and RxTel use the macros defined in ECAT(32/64)MACROS.H

SYBERA Copyright © 2014

### 3.2.3 Structure DATA_DESC

The data fields of the TX / RX process telegram are described by the structure DATA_DESC, which keeps information about item type, data type and data len.

```
typedef struct _DATA_DESC
{
      UCHAR  Item;      //Data Item (e.g. DATA_ITEM_STATUS, DATA_ITEM_VALUE,
      ...)
      UCHAR  Type;      //Data Type (e.g. DATA_TYPE_U8, DATA_TYPE_U16, ...)
      USHORT Len;       //Data Len  (in Bytes)

} DATA_DESC, *PDATA_DESC;
```

The data descriptors may be used to initialize process telegrams with the Library LowLevel Interface (its not required when using the Library HighLevel interface function ShaEcatEnable):

```
__inline void __InitProcessTelegram(PSTATION_INFO pStation)
{
      ULONG LogicalAddr = 0;
      USHORT      DataSize = 0;
      UCHAR Cmd = 0;
      TYPE32      Addr;
      ULONG FmmuIndex[2] = { -1, -1 };
      ULONG i;

      //Get FMMU index for input and output (if available)
      if (pStation->OutDescNum) { FmmuIndex[0] = pStation->OutDescList[0].Fmmu;
}
      if (pStation->InDescNum)   { FmmuIndex[1] = pStation->InDescList[0].Fmmu;
}

      //Loop through all FMMUs
      for (i=0; i<pStation->FmmuNum; i++)
      {
            //Check for input or output FMMU
            if ((i == FmmuIndex[0]) ||
                  (i == FmmuIndex[1]))
            {
                  //Set same logical address for input or output FMMU
                  LogicalAddr = pStation->FmmuList[i].s.LogicalAddr;

                  //Save max. length
                  if (DataSize < pStation->FmmuList[i].s.Length)
                        DataSize = pStation->FmmuList[i].s.Length;
            }
      }

      //Set command, address and len due to descriptors
      if ((pStation->OutDescNum != 0) && (pStation->InDescNum != 0))
```

SYBERA Copyright © 2014

```
            { Cmd = LRW_CMD; DataSize = DataSize; Addr.bit32 = LogicalAddr; }

      if ((pStation->OutDescNum != 0) && (pStation->InDescNum == 0))
            { Cmd = LWR_CMD; DataSize = DataSize; Addr.bit32 = LogicalAddr; }

      if ((pStation->OutDescNum == 0) && (pStation->InDescNum != 0))
            { Cmd = LRD_CMD; DataSize = DataSize; Addr.bit32 = LogicalAddr; }

      if ((pStation->OutDescNum == 0) && (pStation->InDescNum == 0))
            {
                  Cmd = BRD_CMD; DataSize = sizeof(AL_STATUS);
                  Addr.bit16[0] = 0x0000;
                  Addr.bit16[1] = 0x0130;
            }

      //Set cyclic telegram
      __EcatSetCyclicTelegram(
                        &pStation->TxTel,
                        (UCHAR)pStation->Index,
                        Cmd,
                        Addr.bit16[0],
                        Addr.bit16[1],
                        DataSize,
                        NULL,
                        0x0000);

      //Set station update
      pStation->bUpdate = TRUE;
}
```

SYBERA Copyright © 2014

### 3.3 Header File ECAT*(32/64)*MACROS.H

This header file defines all macros required for handling realtime level 2.

This Inline-Macro is to set telegram information:

```
__EcatSetTelegram(__pTel,__index,__cmd,__adp,__ado,__DataSize,__pData,__WorkCnt)

__pTel            Type: PECAT_TELEGRAM          //EtherCAT Telegram
__index           Type: UCHAR                   //Telegram index
__cmd             Type: UCHAR                   //Telegram command
__adp             Type: USHORT                  //Telegram ADP
__ado             Type: USHORT                  //Telegram ADO
__DataSize        Type: ULONG                   //Telegram Data Size
__pData           Type: PUCHAR                  //Telegram Data pointer
__WorkCnt         Type: USHORT                  //Telegram Working Count
```

This Inline-Macro is to set cyclic telegram information:

```
__EcatSetCyclicTelegram(__pTel,__index,__cmd,__adp,__ado,__DataSize,__pData,__Wor
      kCnt)
```

This Inline-Macro is to get telegram information:

```
__EcatGetTelegram(__pTel,__pIndex,__pCmd,__pAdp,__pAdo,__DataSize,__pData,__pWor
      kCnt)

__pTel            Type: PECAT_TELEGRAM          //EtherCAT Telegram
__pIndex          Type: PUCHAR                  //Telegram index
__pCmd            Type: PUCHAR                  //Telegram command
__pAdp            Type: PUSHORT                 //Telegram ADP
__pAdo            Type: PUSHORT                 //Telegram ADO
__DataSize        Type: ULONG                   //Telegram Data bytes to copy
__pData           Type: PUCHAR                  //Telegram Data pointer
__pWorkCnt        Type: PUSHORT                 //Telegram Working Count
```

This Inline-Macro is to copy telegrams:

```
__EcatCpyTelegram(__pDstTel,__pSrcTel)

__pDstTel         Type: PECAT_TELEGRAM
__pSrcTel         Type: PECAT_TELEGRAM
```

SYBERA Copyright © 2014

This Inline-Macro is to get the station pointer due to the physical address:

```
PSTATION_INFO __EcatGetStation(pStationList, StationNum, PhysAddr)

pStationList      Type: PSTATION_INFO
StationNum        Type: ULONG
PhysAddr          Type: USHORT
```

SYBERA Copyright © 2014

### 3.4 Header File ECAT*(32/64)*SDODEF.H

This header file defines structures required for COE communication with the Library LowLevel Interface.

```
typedef union _COE_HDR
{
      UCHAR bytes[1];
      struct
      {
            USHORT      Num         : 9;
            USHORT      Reserved    : 3;
            USHORT      Service     : 4;
      } bits;
} COE_HDR, *PCOE_HDR;


typedef union _SDO_INIT_HDR
{
      UCHAR bytes[1];
      struct
      {
            struct
            {
                  UCHAR SizeIndicator     : 1;
                  UCHAR TransferType      : 1;
                  UCHAR DataSetSize       : 2;
                  UCHAR CompleteAccess    : 1;
                  UCHAR Command           : 3;
            } bits;
            USHORT      Index;
            UCHAR SubIndex;
      } s;
} SDO_INIT_HDR, *PSDO_INIT_HDR;


//*** SDO Legacy Request


typedef union _SDO_LEGACY
{
      UCHAR bytes[1];
      struct
      {
            COE_HDR           CoeHdr;
            SDO_INIT_HDR      SdoHdr;
            TYPE32            Data;
      } s;
} SDO_LEGACY, *PSDO_LEGACY;
```

SYBERA Copyright © 2014

### 3.5 Header File ECAT*(32/64)*SIIDEF.H

This header file defines structures required for EEPROM (SII) Access, as well as parsing SII Category information, when using Library LowLevel interface. The elements are described in the EtherCAT specification.

### 3.6 Header File ECAT*(32/64)*DCDEF.H

This header file defines structures required for Distributed Clock Access, when using Library LowLevel interface. The elements are described in the EtherCAT specification.

SYBERA Copyright © 2014

### 3.7 Debug Log File

The EtherCAT master library provides a buildin log sytem which produces a debug log file called *ECATDBG.LOG*. This file contains all nessecary information of the library sequence.

Sample:

```
ECATCORE -> CreateStationList

ECATCORE -> InitStationList

ECATCORE -> GetStationParams
0: Name:EK1100, Vendor:00000002, ProductCode:044c2c52, RevNum:00110000
1: Name:EL1008, Vendor:00000002, ProductCode:03f03052, RevNum:00100000

ECATCORE -> EcatChangeAllStates
0: Name:EK1100 New State: 1
1: Name:EL1008 New State: 1

ECATCORE -> EcatInitStationAddresses
0: Name:EK1100 PhysAddr: 0x000003e9
1: Name:EL1008 PhysAddr: 0x000003ea

ECATCORE -> EcatInitFmmus
1: Name:EL1008 Transfered FMMU: 0

ECATCORE -> EcatInitSyncManagers
1: Name:EL1008 Transfered SYNCMAN: 0

ECATCORE -> EcatChangeAllStates
0: Name:EK1100 New State: 2
1: Name:EL1008 New State: 2

ECATCORE -> EcatPdoAssignment

ECATCORE -> EcatChangeAllStates
0: Name:EK1100 New State: 4
1: Name:EL1008 New State: 4

ECATCORE -> EcatChangeAllStates
0: Name:EK1100 New State: 8
1: Name:EL1008 New State: 8

ECATCORE -> EcatChangeAllStates
0: Name:EK1100 New State: 1
1: Name:EL1008 New State: 1

ECATCORE -> DestroyStationList
```

SYBERA Copyright © 2014

# 4 EtherCAT Library HighLevel Interface

The header file SHAECATCORE.H defines all required prototypes and parameters of the Ethernet Core Library. In the following all function prototypes will be discussed by samples. Since all platforms have their own syntax and dependencies, therefore the topics for the different platforms are marked as follow:

VC : Visual C and Borland C++ Builder

VB : Visual Basic

DP : Borland Delphi

### 4.1.1 Sha*(32/64)*EcatGetVersion

This function retrieves the version information strings of the EtherCAT Master Library, the Ethernet Core Library, the Ethernet Core Driver, the SHA Dll, the SHA Library and the SHA Driver. The memory for the information strings must be allocated first.

VC
```
ULONG Sha(32/64)EcatGetVersion (PECAT_PARAMS);
```

Sample:

```
//Display version information
ShaEcatGetVersion(&EcatParams);
printf("ECTCORE-DLL : %.2f\nECTCORE-DRV : %.2f\n",
      EcatParams.core_dll_ver / (double)100,
      EcatParams.core_drv_ver / (double)100);

printf("ETHCORE-DLL : %.2f\nETHCORE-DRV : %.2f\n",
      EcatParams.EthParams.core_dll_ver / (double)100,
      EcatParams.EthParams.core_drv_ver / (double)100);

printf("SHA-LIB     : %.2f\nSHA-DRV     : %.2f\n",
      EcatParams.EthParams.sha_lib_ver / (double)100,
      EcatParams.EthParams.sha_drv_ver / (double)100);
```

SYBERA Copyright © 2014

### 4.1.2 Sha*(32/64)*EcatCreate

This function initializes the EtherCAT Realtime and Station Management. On success the returning value is ERROR_SUCCESS, otherwise the returning value corresponds to that with GetLastError().

VC `ULONG Sha(32/64)ECatCreate (PECAT_PARAMS);`

Sample:

```
//Required ECAT parameters
ECAT_PARAMS EcatParams;
EcatParams.FixedAddress = 1001;
EcatParams.LogicalAddress = 0x00010000;
EcatParams.SyncCycles = 20
EcatParams.EthParams.dev_num = 0;
EcatParams.EthParams.period = 100;
EcatParams.EthParams.sched_cnt = 1;
EcatParams.EthParams.fpAppTask = AppTask;

//Enable ECAT realtime core
if (ERROR_SUCCESS == ShaECatCreate(&EcatParams))
{
      //Init global realtime elements
      __pUserStack      = EcatParams.EthParams.pUserStack;
      __pSystemStack    = EcatParams.EthParams.pSystemStack;
      __pUserList       = EcatParams.pUserList;
      __pSystemList     = EcatParams.pSystemList;
      __StationNum      = EcatParams.StationNum;
      __fpEcatEnter     = EcatParams.fpEcatEnter;
      __fpEcatExit      = EcatParams.fpEcatExit;
}
```

Note:
The parameter period is the base sampling rate (e.g. 100µsec) for RX, TX and ERR tasks. Cyclic Ethtercat telegrams will be handled by a synchronizing period:

*EcatParams.EthParams.period * EcatParams.SyncCycles*

(e.g. 100µsec * 20 = 2msec)

SYBERA Copyright © 2014

Note: Library LowLevel Routines implemented by ShaEcatCreate

//Clear CRC Fault Counter and reset devices
Ecat*(32/64)*ResetDevices();
Ecat*(32/64)*CheckErrorCounters(BOOLEAN bReset);

//Get DL and PDI information for each module
Ecat*(32/64)*ReadDlInfo();
Ecat*(32/64)*ReadDlStatus();
Ecat*(32/64)*ReadPDIControl();

//Init DC
Ecat*(32/64)*ReadDcLocalTime();
Ecat*(32/64)*CompDcPropDelay();
Ecat*(32/64)*CompDcOffset();
Ecat*(32/64)*CompDcDrift(PULONG &DriftTimePerMsec);
Ecat*(32/64)*ReadDcSyncInfo();

Note: Logical Addessing Sheme

The EtherCAT Realtime Library provides an integrated logical addressing sheme. Thereby all EtherCAT stations get an logical address due to the following algorithm:

```
for (ULONG i=0; i <StationNum; i++)
for (ULONG FmmuIndex=0; FmmuIndex <StationList[i]->FmmuNum; FmmuIndex ++)
{
      //Increase logical address with gap and alignment
      pStation->LogicalAddr += pStation->FmmuList[FmmuIndex].s.Length;
      pStation->LogicalAddr += 0x10;
      pStation->LogicalAddr = ALIGN_SIZE(LogicalAddr, 0x10);
}
```

SYBERA Copyright © 2014

### 4.1.3 Sha*(32/64)*EcatDestroy

This function closes the EtherCAT communication.

VC `ULONG Sha(32/64)ECatDestroy(PECAT_PARAMS);`

### 4.1.4 Sha*(32/64)*EcatEnable

This function enables the EtherCAT station list and must follow the function ShaEcatCreate.

VC `ULONG Sha(32/64)ECatEnable(PECAT_PARAMS);`

Note: Library LowLevel Routines implemented by this Function

```
EcatChangeAllStates(AL_STATE_INIT)          //Change state to INIT
EcatInitStationAddresses(pParams->PhysAddr) //Set fixed station addresses
EcatInitFmmus(pParams->LogicalAddr)         //Init FMMUs and SYNCMANs
EcatInitSyncManagers()
EcatChangeAllStates(AL_STATE_PRE_OP)        //Change state to PRE OPERATIONAL
EcatPdoAssignment()                         //Init PDO assignment
EcatChangeAllStates(AL_STATE_SAFE_OP)       //Change state to SAFE
                                            //OPERATIONAL
EcatChangeAllStates(AL_STATE_OP)            //Change state to OPERATIONAL
```

### 4.1.5 Sha*(32/64)*EcatDisable

This function disables the EtherCAT station list

VC `ULONG Sha(32/64)ECatDisable(PECAT_PARAMS);`

SYBERA Copyright © 2014

Sample:

```
//*****************************************************************
// This sample demonstrates how to use ETHERCAT Realtime Core
// in Realtime Level2 with Beckhoff modules EK1100, EL2032 and EL1014
//*****************************************************************

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "c:\eth\EthCoreDef.h"
#include "c:\eth\EthMacros.h"
#include "c:\ect\EcatCoreDef.h"
#include "c:\ect\EcatMacros.h"
#include "c:\ect\ShaEcatCore.h"

//Global elements
PETH_STACK        __pUserStack = NULL;    //Ethernet Core Stack (outside
                                          //Realtime)
PETH_STACK        __pSystemStack = NULL;  //Ethernet Core Stack (inside Realtime)
PSTATION_INFO     __pUserList = NULL;     //Station List (outside Realtime)
PSTATION_INFO     __pSystemList = NULL;   //Station List (inside Realtime)
USHORT            __StationNum = 0;       //Number of Stations
FP_ECAT_ENTER     __fpEcatEnter = NULL;   //Function pointer to Wrapper EcatEnter
FP_ECAT_EXIT      __fpEcatExit = NULL;    //Function pointer to Wrapper EcatExit
ULONG             __EcatState = 0;        //Initial Wrapper State
ULONG             __UpdateCnt = 0;        //Station Update Counter
ULONG             __LoopCnt = 0;          //Realtime Cycle Counter
ULONG             __ReadyCnt = 0;         //Ready state counter


void static AppTask(void)
{
      //Check if system memory is present
      if ((!__pSystemStack) ||
            (!__pSystemList))
            return;

      //Call enter wrapper function
      __EcatState = __fpEcatEnter(
                                          __pSystemStack,
                                          __pSystemList,
                                          (USHORT)__StationNum,
                                          NULL);

      //Check operation state and increase ready count
      if (__EcatState == ECAT_STATE_READY) { __ReadyCnt++; }
      else                                 { __ReadyCnt=0; }

      //Check ready count for cycle operation
      if (__ReadyCnt == 1)
      {
            //*********************************
            //Do the logical station operation
            //e.g. toogle output value (station index == 2)
//          __pSystemList[2].TxTel.s.data[0] =
      (__pSystemList[1].TxTel.s.data[2]) ? 0x00 : 0x03;
```

SYBERA Copyright © 2014

```
            //*********************************

            __UpdateCnt++;
      }

      //Call exit function
      __fpEcatExit();

      //Increase loop count
      __LoopCnt++;
}


void main(void)
{
      printf("\n*** EtherCAT Core Realtime Level2 Test ***\n\n");

      //Required ECAT parameters
      ECAT_PARAMS EcatParams;
      EcatParams.PhysAddr = DEFAULT_PHYSICAL_ADDRESS;
      EcatParams.LogicalAddr = DEFAULT_LOGICAL_ADDRESS;
      EcatParams.SyncCycles = 10;
      EcatParams.EthParams.dev_num = 0;
      EcatParams.EthParams.eth_type = ETH_TYPE_ECAT; //Set ethernet frame type
      filter for selected interface
      EcatParams.EthParams.eth_if = ETH_IF_CORE;     //Set filter interface
      (Send
                                                     //all other ethernet frames
                                                     //to socket interface)
      EcatParams.EthParams.period = 200;             //Set realtime period
      [µsec]
      EcatParams.EthParams.sched_cnt = 1;            //Set application task
                                                     //scheduler count (cycle
                                               time =
                                                     //sched_cnt * period)
      EcatParams.EthParams.fpAppTask = AppTask;

      //*****************************
      //Create ECAT realtime core
      //*****************************
      if (ERROR_SUCCESS == ShaEcatCreate(&EcatParams))
      {
            //Init realtime elements
            __pUserStack      = EcatParams.EthParams.pUserStack;
            __pSystemStack    = EcatParams.EthParams.pSystemStack;
            __pUserList       = EcatParams.pUserList;
            __pSystemList     = EcatParams.pSystemList;
            __StationNum      = EcatParams.StationNum;
            __fpEcatEnter     = EcatParams.fpEcatEnter;
            __fpEcatExit      = EcatParams.fpEcatExit;

            //*****************************
            //Enable Stations
            //*****************************
            if (ERROR_SUCCESS == ShaEcatEnable(&EcatParams))
            {
                  //Display remain time
```

SYBERA Copyright © 2014

```
                printf("\nRemain Time: %i\n\n",
      EcatParams.EthParams.remain_time);

                //Display station information
                for (int i=0; i<__StationNum; i++)
                      printf("Station: %i, Name: %6s\n", i,
      __pUserList[i].szName);

                //Do a check loop
                printf("\nPress any key ...\n");
                while (!kbhit())
                {
                      //Display TX and RX information
                      printf("Loop Count: %i, Update Count: %i\r",
                            __LoopCnt, __UpdateCnt);

                      //Do some delay
                      Sleep(100);
                }

                //Disable Stations
                ShaEcatDisable(&EcatParams);
          }
          //Destroy ECAT core
          ShaEcatDestroy(&EcatParams);
      }
}
```

SYBERA Copyright © 2014

# 5 Realtime Operation

After changing a device into operational state, the cyclic operation is active. The realtime task is decorated by the Realtime EtherCAT Wrapper functions:

```
typedef ULONG   (__cdecl *FP_ECAT_ENTER)(PETH_STACK, PSTATION_INFO, SHORT);
typedef VOID    (__cdecl *FP_ECAT_EXIT) (VOID);
```

These wrapper functions are used to manage the realtime EtherCAT station management, like ethernet frame update, error handling, synchronisation and stack management. Since the Application task is running with a sampling period (e.g. 200µsec), the wrapper each period returns one of the following states:

```
//Define ECAT states
enum _ECAT_STATE
{
      ECAT_STATE_INIT = 0,    //Initial state
      ECAT_STATE_UPDATE,      //Update still in progress
      ECAT_STATE_READY,       //All stations are updated
      ECAT_STATE_ERROR        //An update error occured
};
```

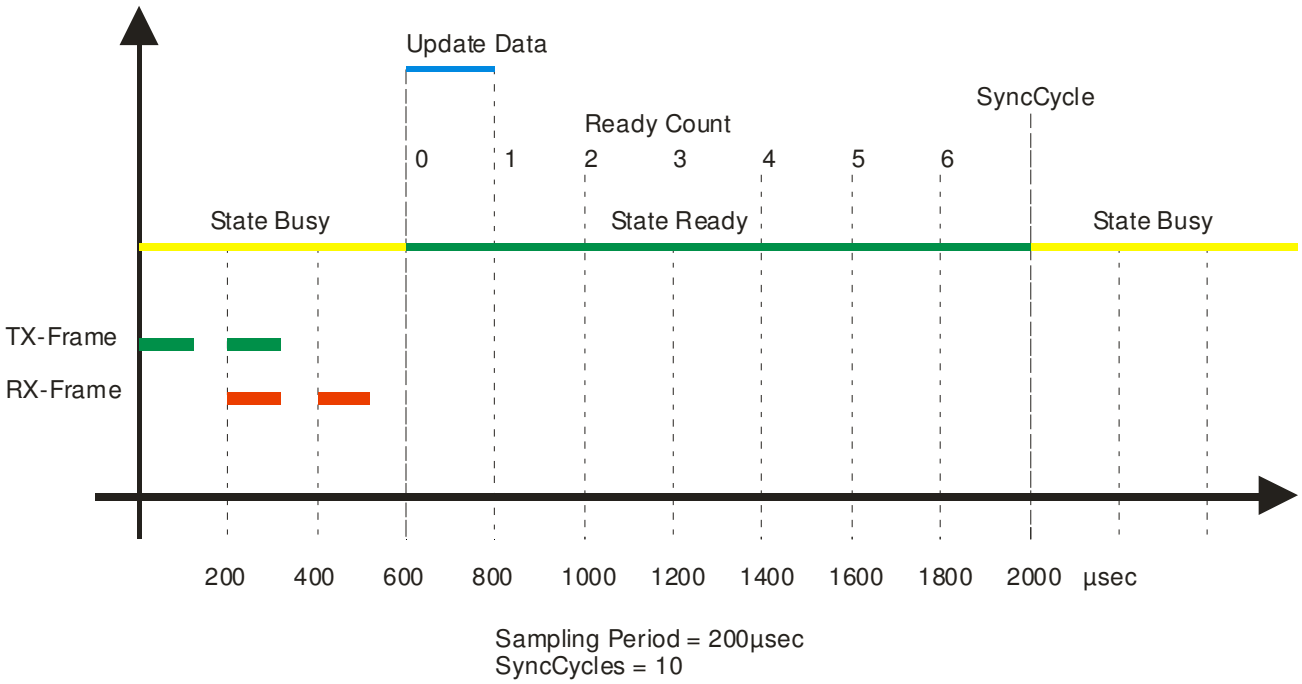
SYBERA Copyright © 2014

The Wrapper Functions require as parameters the Ethernet Stack pointer (e.g.__pSystemStack), the station list pointer (e.g.__pSystemList) and the number of stations (e.g. __StationNum). These parameters and others are returned when initializing the EtherCAT Realtime Library and are set as global elements.

```
//Declare global elements
PETH_STACK     __pUserStack = NULL;           //Ethernet Core Stack
                                              //(used outside Realtime Task)
PETH_STACK     __pSystemStack = NULL;         //Ethernet Core Stack
                                              //(used inside Realtime Task)
PSTATION_INFO __pUserList = NULL;             //Station List
                                              //(used outside Realtime Task)
PSTATION_INFO __pSystemList = NULL;           //Station List
                                              //(used inside Realtime Task)
USHORT         __StationNum = 0;              //Number of Stations
FP_ECAT_ENTER __fpEcatEnter = NULL;           //Function pointer to Wrapper
EcatEnter
FP_ECAT_EXIT   __fpEcatExit = NULL;           //Function pointer to Wrapper
EcatExit
ULONG          __EcatState = ECAT_STATE_STOP; //Initial Wrapper State
ULONG          __UpdateCnt = 0;               //Station Update Counter
ULONG          __LoopCnt = 0;                 //Realtime Loop Counter
ULONG          __ReadyCnt = 0;                //Ready state counter
```

SYBERA Copyright © 2014

The realtime task returns the EtherCAT wrapper state with each sampling period (e.g. 200 µsec). When the wrapper indicate the state ECAT_STATE_READY it means, that all stations are updated. Within one synchronisation cycle (e.g. 2msec), the data should updated just once. Since in the following sample the state ECAT_STATE_READY would last 7 sampling periods, its useful to keep track by a ready counter and update the data just once within one synchronisation cycle:




Sampling Period = 200µsec
SyncCycles = 10

SYBERA Copyright © 2014

```
void static AppTask(void)
{
      //Check if system memory is present
      if ((!__pSystemStack) ||
          (!__pSystemList))
            return;

      //Call enter wrapper function
      __EcatState = __fpEcatEnter(
                              __pSystemStack,
                              __pSystemList,
                              (USHORT)__StationNum,
                              NULL);

      //Check operation state and increase ready count
      if (__EcatState == ECAT_STATE_READY) { __ReadyCnt++; }
      else                                 { __ReadyCnt=0; }

      //Check ready count for cycle operation
      if (__ReadyCnt == 1)
      {
            //DBG_INITIAL_BREAK();

            //*********************************
            //Do the logical station operation
            //e.g. toogle output value (station index == 2)
//          __pSystemList[2].TxTel.s.data[0] =
(__pSystemList[1].TxTel.s.data[2]) ? 0x00 : 0x03;
            //*********************************

            __UpdateCnt++;
      }

      //Call exit function
      __fpEcatExit();

      //Increase loop count
      __LoopCnt++;
}
```

SYBERA Copyright © 2014

# 6 EtherCAT Library LowLevel Interface

The EtherCAT Library LowLevel Interface provides all functions to control Slave devices in detail. Several LowLevel Interface groups are provided by this library.

## 6.1 EtherCAT LowLevel Command Functions

The EtherCAT realtime library allows controlling EtherCAT at low level. Therefore several commands are exported as low level functions.

### 6.1.1 Send EtherCAT Command

This is an universal function for sending EtherCAT commands

```
ULONG Result = Ecat(32/64)SendCommand(
                                UCHAR       Cmd,
                                USHORT      Adp,
                                USHORT      Ado,
                                USHORT      DataSize,
                                PUCHAR      pData)
```

Sample:

```
//Send ethercat command
ULONG Result = EcatSendCommand(APWR_CMD, 0xFFFE, 0x120, 2, (PUCHAR)"\x01\x00");
```

### 6.1.2 Reset Devices

This command proceeds following actions:

- Empty any pending ethercat frames
- Reset DL control : BWR Offs 0x101
- Clear FMMUs : BWR Offs 0x600 - 0x6FF
- Clear SyncManager : BWR Offs 0x800 - 0x8FF
- Write to SystemTime : BWR Offs 0x910
- Write to Cycle Operation Start Time : BWR Offs 0x981
- Write to : BWR Offs 0x930
- Set event mask : BWR Offs 0x934

```
ULONG Result = Ecat(32/64)ResetDevices(void);
```

SYBERA Copyright © 2014

### 6.1.3 Clear Error Counters

Read or Reset RX Error Counter : BWR Offs 0x300 - 0x303

```
ULONG Result = Ecat(32/64)CheckErrorCounters(BOOLEAN bReset);
```

### 6.1.4 Read DL Information

Read DL information into station list

```
ULONG Result = ECat(32/64)ReadDlInfo(void);
```

### 6.1.5 Read DL Status

Read DL Status information into station list

```
ULONG Result = Ecat(32/64)ReadDlStatus(void);
```

### 6.1.6 Read/Write DL Control

Read/Write DL Control information into station list

```
ULONG Result = Ecat(32/64)CheckDlControl(BOOLEAN bWrite);
```

### 6.1.7 Read PDI Control

Read PDI Control information into station list

```
ULONG Result = Ecat(32/64)ReadPDIControl(void);
```

### 6.1.8 Read PDI Configuration

Read PDI Configuration information into station list

```
ULONG Result = Ecat(32/64)ReadPDIConfig(void);
```

SYBERA Copyright © 2014

### 6.1.9 Init Station Addresses

Initialize all station physical addresses, beginning from the physical start address

```
ULONG Result = Ecat(32/64)InitStationAddresses(USHORT PhysStartAddress);
```

### 6.1.10 Init Alias Addresses

Initialize all station alias addresses (requires station element AliasAddr set before)

```
ULONG Result = Ecat(32/64)InitAliasAddresses(void);
```

### 6.1.11 Configure SYNC Management

Initialize all SYNC Managers of all stations due to the native parameter file, or the EEPROM information (the SYCMAN list of the station must be set before).

```
ULONG Result = Ecat(32/64)InitSyncManagers(void);
```

### 6.1.12 Configure FMMU Management

Initialize all FMMU Managers of all stations due to the native parameter file, or the EEPROM information (the FMMU list of the station must be set before).

```
ULONG Result = Ecat(32/64)InitFmmus (void);
```

SYBERA Copyright © 2014

### 6.1.13 Configure PDO Assignment

Initialize all PDOs of all stations due to the native parameter file, or the EEPROM information (the SDO list of the station must be set before).

```
ULONG Result = Ecat(32/64)PdoAssignment (void);
```

This command proceeds following actions:

- Check mailbox for pending response
- Write COE command to mailbox
- Read COE command from mailbox
- Check SDO response

### 6.1.14 Watchdog Enable

Enables/Disables all watchdog controls of the station list

```
ULONG ECatWatchdogEnable(BOOLEAN bEnable)
```

SYBERA Copyright © 2014

## 6.2 EtherCAT LowLevel State Functions

The EtherCAT realtime library allows managing the EtherCAT states of all enabled stations at low level. Therefore several functions are exported as low level state functions.

### 6.2.1 Read AL Status

Read AL status of all stations: APRD Offs 0x130

```
ULONG Result = ECat(32/64)ReadAlStatus(void);
```

### 6.2.2 Change All States

Change of all station states. The addressing scheme depends on the current state (APWR/APRD at AL_STATE_INIT, else FPWR/FPRD). Thus, changing states requires set of station addresses before.

```
ULONG Result = ECat(32/64)ChangeAllStates(UCHAR State);
```

### 6.2.3 Change State By Node Address

Change of a single station state. The addressing scheme depends on the current state (APWR/APRD at AL_STATE_INIT, else FPWR/FPRD). Thus, changing states requires set of station addresses before.

```
ULONG Result = ECat(32/64)ChangeStatesByNodeAddress(
                                        UCHAR  State,
                                        USHORT StationAddress);
```

SYBERA Copyright © 2014

Sample:

```
//Change state to INIT
if (ERROR_SUCCESS == EcatChangeAllStates(AL_STATE_INIT))
{
      //Set fixed station addresses and
      //Init FMMUs and SYNCMANs
      if (ERROR_SUCCESS == EcatInitStationAddresses(EcatParams.PhysAddr))
      if (ERROR_SUCCESS == EcatInitFmmus(EcatParams.LogicalAddr))
      if (ERROR_SUCCESS == EcatInitSyncManagers())

      //Change state to PRE OPERATIONAL
      if (ERROR_SUCCESS == EcatChangeAllStates(AL_STATE_PRE_OP))
      {
            //Init PDO assignment and
            //Change state to SAFE OPERATIONAL
            if (ERROR_SUCCESS == EcatPdoAssignment())
            if (ERROR_SUCCESS == EcatChangeAllStates(AL_STATE_SAFE_OP))
            {
                  //Change state to OPERATIONAL
                  if (ERROR_SUCCESS == EcatChangeAllStates(AL_STATE_OP))
                  {
                        //Init process telegrams
                        InitProcessTelegrams();
                        …
```

SYBERA Copyright © 2014

## 6.3 EtherCAT LowLevel COE Functions

The EtherCAT realtime library allows COE-SDO communication with corresponding modules at low level. Therefore several functions are exported as low level SDO functions.

### 6.3.1 Initiate SDO Download Expedited Request

This function initiates a SDO Download Expedited Request

```
ULONG SHAAPI Ecat(32/64)SdoInitDownloadReq(
                              PSTATION_INFO pStation,
                              USHORT SdoIndex,
                              UCHAR SdoSubIndex,
                              ULONG SdoDataSize,
                              PUCHAR pSdoData)
```

### 6.3.2 Initiate SDO Download Expedited Response

This function initiates a SDO Download Expedited Response

```
ULONG Ecat(32/64)SdoInitDownloadResp(PSTATION_INFO pStation);
```

### 6.3.3 Initiate SDO Upload Expedited Request

This function initiates a SDO Upload Expedited Request

```
ULONG Ecat(32/64)SdoInitUploadReq(
                        PSTATION_INFO pStation,
                        USHORT SdoIndex,
                        UCHAR  SdoSubIndex);
```

### 6.3.4 Initiate SDO Download Expedited Response

This function initiates a SDO Download Expedited Response

```
ULONG Ecat(32/64)SdoInitUploadResp(
                        PSTATION_INFO pStation,
                        PULONG  pSdoDataSize,
                        PUCHAR* ppSdoData)
```

SYBERA Copyright © 2014

Sample:

```
//Reset SDO data
memset(pCmd, 0, CmdSize);

//Set CoE header
PCOE_HDR pCoeHdr = (PCOE_HDR)pCmd;
pCoeHdr->bits.Num = 0;
pCoeHdr->bits.Service = COE_SERVICE_SDOREQ;

//Set SDO Init header (SDO Init Download Expedited Request)
PSDO_INIT_HDR pSdoInitHdr = (PSDO_INIT_HDR)&pCmd[sizeof(COE_HDR)];
pSdoInitHdr->s.bits.SizeIndicator = TRUE;
pSdoInitHdr->s.bits.TransferType = TRUE;
pSdoInitHdr->s.bits.DataSetSize = DataSetSize;
pSdoInitHdr->s.bits.CompleteAccess = FALSE;
pSdoInitHdr->s.bits.Command = SDO_INIT_DOWNLOAD_REQ;
pSdoInitHdr->s.Index = SdoIndex;
pSdoInitHdr->s.SubIndex = SdoSubIndex;

//Set SDO data
memcpy(
(PUCHAR)&pCmd[sizeof(COE_HDR) + sizeof(SDO_INIT_HDR)],
pSdoData,
SdoDataSize);

//Check mailbox for pending response
EcatMailboxCheck(pStation);

//Write COE command from mailbox
ULONG dwResult = EcatMailboxWrite(pStation, pCmd, CmdSize, MBX_TYPE_COE);
```

SYBERA Copyright © 2014

## 6.4 EtherCAT LowLevel Mailbox Functions

The EtherCAT realtime library allows mailbox communication with corresponding modules at low level. Therefore several functions are exported as low level mailbox functions.

### 6.4.1 Write command to mailbox

This function writes to a mailbox

```
ULONG Result = Ecat(32/64)MailboxWrite(
                        PSTATION_INFO pStation,
                        PUCHAR pData,
                        USHORT DataSize,
                        UCHAR MailboxType)
```

### 6.4.2 Read command from mailbox

This function reads from a mailbox

```
ULONG Result = Ecat(32/64)MailboxRead(
                        PSTATION_INFO pStation,
                        PUCHAR pData)
```

### 6.4.3 Check mailbox for pending response

This function checks a mailbox for pending response

```
ULONG Result = Ecat(32/64)MailboxCheck(PSTATION_INFO pStation)
```

SYBERA Copyright © 2014

## 6.5 EtherCAT LowLevel EEPROM Functions

The EtherCAT realtime library allows EEPROM (SII) access to the corresponding modules at low level. Additionally the library provides parser functions for SII Category data. Therefore several functions are exported as low level functions.

### 6.5.1 Read SII Data

This function reads a range of SII data, due to a given offset, into a given data buffer

```
ULONG Result = Ecat(32/64)SiiRead(
                        PSTATION_INFO pStation,
                        PUCHAR pData,
                        USHORT DataSize,
                        USHORT Offs)
```

### 6.5.2 Write SII Data

This function writes a data buffer into the SII area, due to a given offset

```
ULONG Result = Ecat(32/64)SiiWrite(
                        PSTATION_INFO pStation,
                        PUCHAR pData,
                        USHORT DataSize,
                        USHORT Offs)
```

### 6.5.3 Reload SII Data

This function reloads the device with EEPROM information, due to a given offset

```
ULONG Result = Ecat(32/64)SiiReload(
                        PSTATION_INFO pStation,
                        USHORT DataSize,
                        USHORT Offs)
```

SYBERA Copyright © 2014

### 6.5.4 Get Category String

This function searches inside the SII area for a general information due to a given index.

```
ULONG SHAAPI Ecat(32/64)GetCategoryGeneral(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pGeneral)
```

### 6.5.5 Get Category String

This function searches inside the SII area for a string due to a given index. If the string pointer is NULL, the function returns the number of strings inside the SII area.

```
ULONG Result = Ecat(32/64)GetCategoryString(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                char* pszStr,
                                ULONG StrIndex)
```

### 6.5.6 Get Category SYNC Manager

This function searches inside the SII area for a SYNC Manager due to a given index. If the SYNC Manager pointer is NULL, the function returns the number of SYNC Managers inside the SII area.

```
ULONG Result = Ecat(32/64)GetCategorySyncman(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pSyncman,
                                ULONG SyncmanIndex)
```

### 6.5.7 Get Category FMMU Manager

This function searches inside the SII area for a FMMU Manager due to a given index. If the FMMU Manager pointer is NULL, the function returns the number of FMMU Manager inside the SII area.

```
ULONG Result = Ecat(32/64)GetCategoryFmmu(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pFmmu,
                                ULONG FmmuIndex)
```

SYBERA Copyright © 2014

### 6.5.8 Get Category PDOs

This function searches inside the SII area for a PDOs due to a given index. If the PDO pointer is NULL, the function returns the number of PDOs inside the SII area.

```
ULONG Result = Ecat(32/64)GetCategoryPdo(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pPdo,
                                ULONG PdoIndex,
                                BOOLEAN bTxPdo)
```

Sample:

```
//Read category area
if (ERROR_SUCCESS == EcatSiiRead(
                                m_pStation,
                                m_CatArea, MIN_CAT_AREA_SIZE,
                                sizeof(SII_AREA_HDR)))
{
     //Get general device information
     EcatGetCategoryGeneral(CatArea, MIN_CAT_AREA_SIZE, (PUCHAR)&CatGeneral);

     //Get FMMU category
     int FmmuNum = EcatGetCategoryFmmu(CatArea, MIN_CAT_AREA_SIZE, NULL, -1);
     for (int i=0; i<FmmuNum; i++)
            EcatGetCategoryFmmu(
                        CatArea, MIN_CAT_AREA_SIZE,
                        (PUCHAR)& FmmuList[i], i);

     //Get SYNCMAN categories
     int SyncmanNum = EcatGetCategorySyncman(CatArea, MIN_CAT_AREA_SIZE, NULL, -
     1);
     for (int i=0; i<SyncmanNum; i++)
            EcatGetCategorySyncman(
                        CatArea, MIN_CAT_AREA_SIZE,
                       (PUCHAR)&SyncmanList[i], i);

     //Get PDO categories
     int PdoNum = EcatGetCategoryPdo(CatArea, MIN_CAT_AREA_SIZE, NULL, -1,
     TRUE);
     for (int i=0; i<PdoNum; i++)
            EcatGetCategoryPdo(
                        CatArea, MIN_CAT_AREA_SIZE,
                        (PUCHAR)&pTxPdoList[i], i, TRUE);

     //Get PDO categories
     int PdoNum = EcatGetCategoryPdo(CatArea, MIN_CAT_AREA_SIZE, NULL, -1,
     FALSE);
     for (int i=0; i<PdoNum; i++)
            EcatGetCategoryPdo(
                        CatArea, MIN_CAT_AREA_SIZE,
                        (PUCHAR)&pRxPdoList[i], i, FALSE);
```

SYBERA Copyright © 2014

## 6.6 EtherCAT LowLevel Distributed Clock Functions

The EtherCAT realtime library provides functions for propagation delay compensation, sytem time offset compensation and drift compensation. Additionaly DC sync control can be managed. Therefore several functions are exported as low level functions.

### 6.6.1 DC Local Time

This function latches out the local time of all stations.

```
ULONG Result = Ecat(32/64)ReadDcLocalTime(VOID);
```

### 6.6.2 DC Propagation Delay Compensation

This function compensates the propagation delay for the stations relations

```
ULONG Result = Ecat(32/64)CompDcPropDelay(VOID);
```

### 6.6.3 DC Offset Compensation

This function compensates the offset of station local time and the reference local time (first DC slave)

```
ULONG Result = Ecat(32/64)CompDcOffset(VOID);
```

### 6.6.4 DC Drift Compensation

This function compensates the static clock drift between reference clock (first DC slave) and all further DC clocks, and returns the drifttime per msec (in nsec units) of the master local time. This allows to keep track on the reference clock.

```
ULONG Result = Ecat(32/64)CompDcOffset(PULONG pDriftTimePerMsec);
```

### 6.6.5 Read DC Cyclic Control

This function reads the complete DC_SYNC_INFO structure for further DC processing

```
ULONG Result = Ecat(32/64)ReadDcSyncInfo (VOID);
```

SYBERA Copyright © 2014

### 6.6.6 DC Sync Control

This function enables the synchronisation output signal, due to the DC settings.

```
ULONG SHAAPI Ecat32SyncControl(
                        PSTATION_INFO pStation,
                        ULONG Sync0CycleTime,
                        ULONG Sync1CycleTime,
                        ULONG Sync0CycleShift,
                        ULONG Sync1CycleShift,
                        BOOLEAN bSync0Pulse,
                        BOOLEAN bSync1Pulse,
                        BOOLEAN bSyncPdiCtrl)
```

Sample:

```
      Ecat32SyncControl(
                  &__pUserList[i],
                  Period * SyncCycles * 1000,   //Sync0 cycle time [nsec]
                  0,                //Sync1 cycle time [nsec]
                  20*1000,          //Sync0 cycle shift [nsec]
                  0,                //Sync1 cycle shift [nsec]
                  TRUE,             //Sync0 pulse flag
                  FALSE,            //Sync1 pulse flag
                  FALSE);           //Sync PDI control
```

Note:
The first DC slave in the network line serves as reference clock.

SYBERA Copyright © 2014

# 7 Device Configuration

Usualy device information is provided by a corresponding XML configuration file. Since the development of software with the EtherCAT Master Library has special needs for programming, the XML file must be parsed and translated into a native format. Therefore the EtherCAT Master Library provides a configuration file called **ECATDEVICE.PAR**, whitch is located in the directory **windows\system32** after installation. The ECATDEVICE.PAR is a text based file with sections for Product Code, Name, SYNC Manager, FMMU Manager, SDO and Data Description. A new device description must start with the signature "**>>>**"

Sample:

```
>>> ***** 09/15/10 14:56:37 *****

[NAME]
EL3102
[VENDOR]
00000002
[CODE]
0c1e3052
[REVISION]
00100000
[SYNCMAN]
00 10 80 00 26 00 01 00
80 10 80 00 22 00 01 00
00 11 00 00 04 00 00 00
80 11 06 00 20 00 01 00
[FMMU]
00 00 00 00 06 00 00 07 80 11 00 01 01 00 00 00
00 00 00 00 01 00 00 00 0d 08 00 01 01 00 00 00
[SDO]
00 20 2f 13 1c 00 00 00 00 00
00 20 2b 13 1c 01 00 1a 00 00
00 20 2b 13 1c 02 01 1a 00 00
00 20 2f 13 1c 00 02 00 00 00
[OUTPUT]
[INPUT]
02 01 01 00 00
02 06 02 00 00
02 01 01 00 00
02 06 02 00 00
```

SYBERA Copyright © 2014

Note: With newer devices the configuration is stored insed the EEPROM. The EtherCAT Master Library is able to configure the devices by parsing the EEPROM information, even without XML file or Native file. But without using the configuration file, the configuration time increases by parsing EEPROM information. The Software **ECATVERIFY** parses XML information and EEPROM information and converts it into the native format and gives additional help for configuration.

### 7.1 Section [NAME]

This section contains the name of the device:

```
[NAME]
EL3102
```

### 7.2 Section [VENDOR]

This section contains the vendor ID of the device:

```
[VENDOR]
00000002
```

### 7.3 Section [CODE]

This section contains the product code of the device:

```
[CODE]
0C1E3052
```

### 7.4 Section [REVISION]

This section contains the revision number of the device:

```
[CODE]
00100000
```

SYBERA Copyright © 2014

## 7.5 Section [SYNCMAN]

This section contains the binary data for the synchronisation manager of the device:

```
[SYNCMAN]
00 18 F6 00 26 00 01 00
F6 18 F6 00 22 00 01 00
00 10 00 00 24 00 00 00
00 11 06 00 20 00 01 00
```

Meaning:

```
| Addr   |  Len   |Cntr    |ChEn
                     |Stat    |Res
|        |        |    |    |    |    |
 00   18   F6   00   26   00   01   00     <- SYNMAN0
 F6   18   F6   00   22   00   01   00     <- SYNMAN1
 00   10   00   00   24   00   00   00     <- SYNMAN2
 00   11   06   00   20   00   01   00     <- SYNMAN3
```

| Parameter | relative address (offset) | Data type | Access type | Access type PDI | Value/description |
|---|---|---|---|---|---|
| Physical start address | 0x0000 | WORD | RW | R | |
| Length | 0x0002 | WORD | RW | R | |
| Buffer type | 0x0004 | Unsigned2 | RW | R | 0x00: buffered<br>0x02: mailbox |
| Direction | 0x0004 | Unsigned2 | RW | R | 0x00: area shall be read from the master<br>0x01: area shall be written by the master |
| reserved | 0x0004 | Unsigned1 | RW | R | 0x00 |
| DLS-user event enable | 0x0004 | Unsigned1 | RW | R | 0x00: DLS-user event is not active<br>0x01: DLS-user event is active (when area was accessed and is no longer locked) |
| Watchdog enable | 0x0004 | Unsigned1 | RW | R | 0x00: watchdog disabled<br>0x01: watchdog enabled |
| reserved | 0x0004 | Unsigned1 | RW | R | 0x00 |
| Write event | 0x0005 | Unsigned1 | R | R | 0x00: no write event<br>0x01: write event |
| Read event | 0x0005 | Unsigned1 | R | R | 0x00: no read event<br>0x01: read event |

SYBERA Copyright © 2014

| | | | | | |
|---|---|---|---|---|---|
| reserved | 0x0005 | unsigned1 | R | R | 0x00 |
| Mailbox state | 0x0005 | Unsigned1 | R | R | 0x00: mailbox empty<br>0x01: mailbox full |
| Buffered state | 0x0005 | Unsigned2 | R | R | 0x00: first buffer<br>0x01: second buffer<br>0x02: third buffer<br>0x03: buffer locked |
| reserved | 0x0005 | Unsigned2 | R | R | 0x00 |
| Channel enable | 0x0006 | Unsigned1 | RW | R | 0x00: channel disabled<br>0x01: channel enabled |
| Repeat | 0x0006 | Unsigned1 | RW | R | |
| reserved | 0x0006 | Unsigned4 | RW | R | 0x00 |
| DC Event 0 with Bus write | 0x0006 | Unsigned1 | RW | R | 0x00: no Event<br>0x01: DC Event if master writes complete buffer |
| DC Event 0 with local write | 0x0006 | Unsigned1 | RW ↔ | R | 0x00: no Event<br>0x01: DC Event if DL-user writes complete buffer |
| Channel enable PDI | 0x0007 | Unsigned1 | R | RW | 0x00: channel disabled<br>0x01: channel enabled |
| RepeatAck | 0x0007 | Unsigned1 | R | RW | shall follow repeat after data recovery |
| reserved | 0x0007 | Unsigned6 | R | RW | 0x00 |

SYBERA Copyright © 2014

## 7.6 Section [FMMU]

This section contains the binary data for the FMMU manager of the device:

```
[FMMU]
06 00 00 00 01 00 00 00 0D 08 00 01 01 00 00 00
00 00 00 00 06 00 00 07 00 11 00 01 01 00 00 00
```

Meaning:

```
| LogAddr(Offs) |  Len  |LogStartBit    |PhysStartBit
                        |LogEndBit      |RdWrEnable
                            |PhysAddr       |ChEnable
|               |       |   |   |       |   |   |             |
 00  00  00  00  01  00  00  00  0D  08  00  01  01  00  00  00 <- FMMU0
 00  00  00  00  06  00  00  07  00  11  00  01  01  00  00  00 <- FMMU1
```

| Parameter | relative address (offset) | Data type | Access type | Access type PDI | Value/description |
|---|---|---|---|---|---|
| Logical start address | 0x0000 | DWORD | RW | R | |
| Length | 0x0004 | WORD | RW | R | |
| Logical start bit | 0x0006 | Unsigned3 | RW | R | |
| reserved | 0x0006 | Unsigned5 | RW | R | 0x00 |
| Logical end bit | 0x0007 | Unsigned3 | RW | R | |
| reserved | 0x0007 | Unsigned5 | RW | R | 0x00 |
| Physical start address | 0x0008 | WORD | RW | R | |
| Physical start bit | 0x000A | Unsigned3 | RW | R | |
| reserved | 0x000A | Unsigned5 | RW | R | 0x00 |
| Read enable | 0x000B | Unsigned1 | RW | R | 0x00: entity will be ignored for read service<br>0x01: entity will be used for read service |
| Write enable | 0x000B | Unsigned1 | RW | R | 0x00: entity will be ignored for write service<br>0x01: entity will be used for write service |
| reserved | 0x000B | Unsigned6 | RW | R | 0x00 |
| Enable | 0x000C | Unsigned1 | RW | R | 0x00: entity not active<br>0x01: entity active |
| reserved | 0x000C | Unsigned15 | RW | R | 0x0000 |
| reserved | 0x000E | WORD | RW | R | 0x0000 |

SYBERA Copyright © 2014

### 7.7 Section [SDO]

This section contains the binary SDO data of the device:

```
[SDO]
00 20 2F 12 1C 00 00 00 00 00
00 20 2F 13 1C 00 00 00 00 00
00 20 2B 13 1C 01 00 1A 00 00
00 20 2B 13 1C 02 01 1A 00 00
00 20 2F 13 1C 00 02 00 00 00
```

Meaning:

```
|NumServ|Cmd|Index  |SubIndex
                      |Data
|       |   |       |   |               |
 00  20  2F  12  1C  00  00  00  00  00 <- COE Cmd0
 00  20  2F  13  1C  00  00  00  00  00 <- COE Cmd1
 00  20  2B  13  1C  01  00  1A  00  00 <- COE Cmd2
 00  20  2B  13  1C  02  01  1A  00  00 <- COE Cmd3
 00  20  2F  13  1C  00  02  00  00  00 <- COE Cmd4
```

SYBERA Copyright © 2014

SDO Header Word and Command Byte

| Frame part | Data Field | Data Type | Value/Description |
|---|---|---|---|
| CANopen Header | Number | Unsigned9 | 0x00 |
| | Reserved | Unsigned3 | 0x00 |
| | Service | Unsigned4 | 0x02: SDO Request |
| SDO | Size Indicator | Unsigned1 | 0x00: size of Data (1..4) unspecified<br>0x01: size of Data in Data Set Size specified |
| | Transfer Type | Unsigned1 | 0x01: Expedited transfer |
| | Data Set Size | Unsigned2 | 0x00: 4 Octet Data<br>0x01: 3 Octet Data<br>0x02: 2 Octet Data<br>0x03: 1 Octet Data |
| | Complete Access | Unsigned1 | 0x00 |
| | Command | Unsigned3 | 0x01: Initiate Download Request |

Sample:

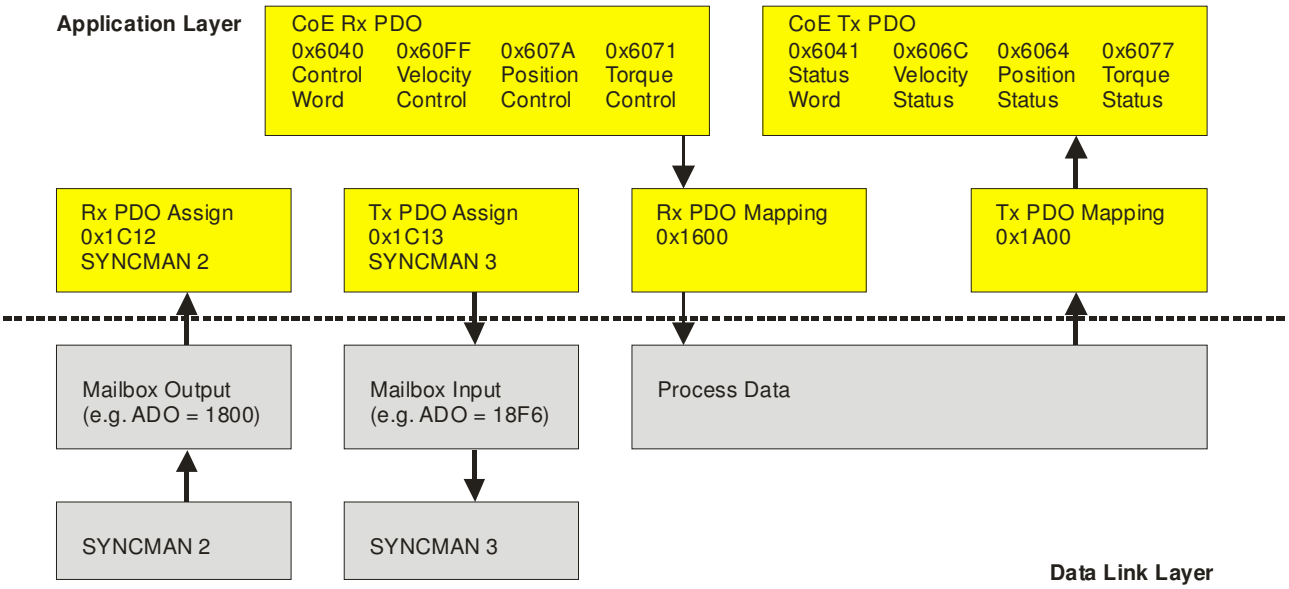
COE Header 2000h : SDO Request
SDO Cmd 2Fh : Data in Data Set Size, exp. Transfer, 1 Oct. Data, Download Req.
Index 1C10h : Sync Manager 0 PDO Assignment (UNSIGNED16)
Index 1C11h : Sync Manager 1 PDO Assignment (UNSIGNED16)
Index 1C12h : Sync Manager 2 PDO Assignment (UNSIGNED16)
Index 1C13h : Sync Manager 3 PDO Assignment (UNSIGNED16)

SYBERA Copyright © 2014

### 7.7.1 PDO Mapping

The PDO mapping allows to assign desired function data to the EtherCAT telegram. The PDO mapping is tunneled via SDO (Service Data Objects).

*PDO mapping by DS402*




| **RxPDO** | **Object** | **Typ** |
|---|---|---|
| 1 | 6040 | Steuerwort |
| 2 | 6060 | Betriebsarten |
| 3 | 607A | Positionssollwert |
| 4 | 60FF | Geschwindigkeitssollwert |
| 5 | 6071 | Drehmomentsollwert |
| … | | |
| **TxPDO** | **Object** | **Typ** |
| 1 | 6041 | Statuswort |
| 2 | 6061 | Betriebsarten |
| 3 | 6064 | Positionsistwert |
| 4 | 606C | Geschwindigkeitsistwert |
| 5 | 6077 | Drehmomentistwert |
| … | | |

SYBERA Copyright © 2014

### 7.8 Section [OUTPUT] / [INPUT]

This section contains the output/input data description of the device:

```
[OUTPUT]
01 01 01 00 00
02 02 02 00 00
01 01 01 00 00
02 02 02 00 00
03 02 02 00 00
03 02 02 00 00
```

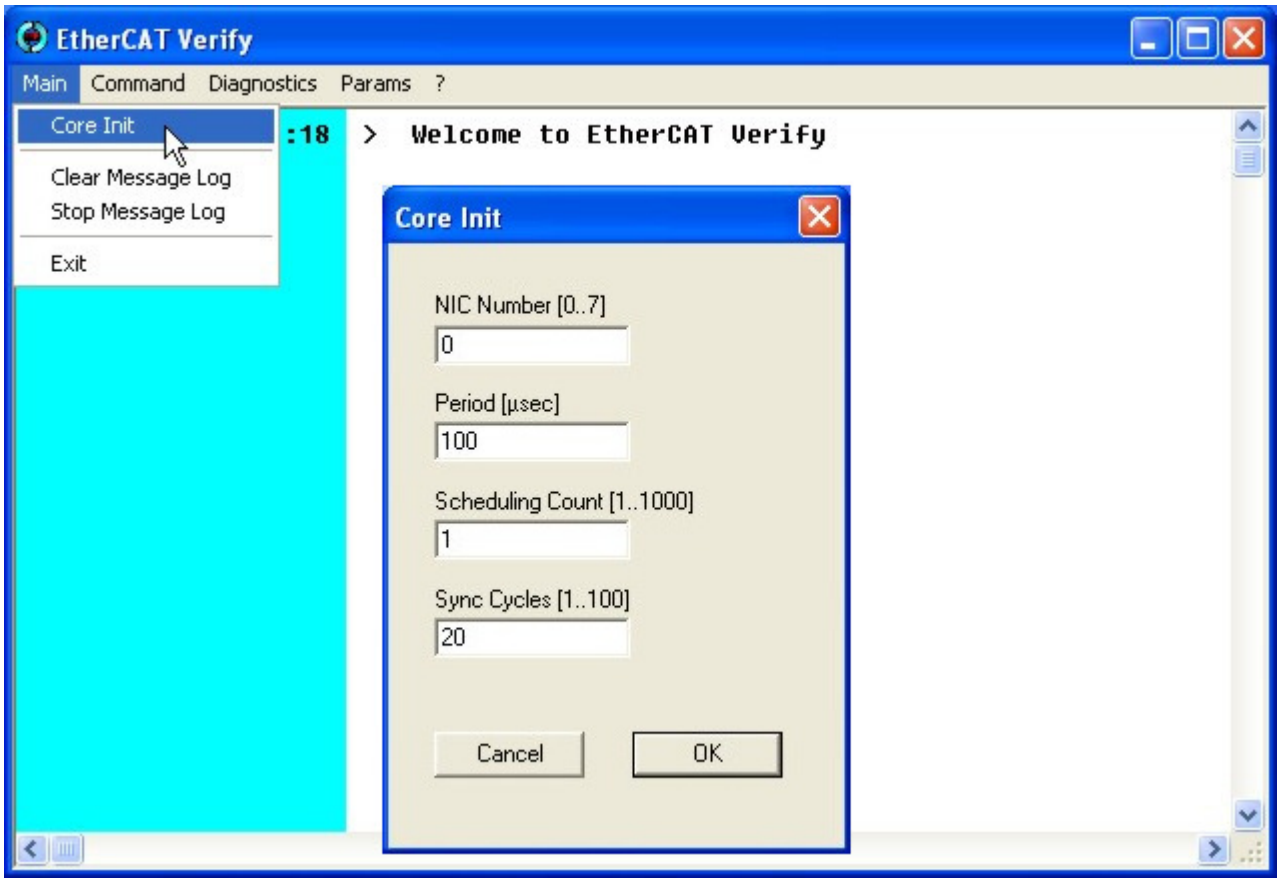
Meaning (see also ECATCOREDEF.H):

```
Item Type  (01 : DATA_ITEM_STATUS)
           (02 : DATA_ITEM_VALUE)
           (03 : DATA_ITEM_SCALE)
           (04 : DATA_ITEM_DIAG)
           (05 : DATA_ITEM_NAME)

Data Type  (01 : DATA_TYPE_U8)
           (02 : DATA_TYPE_U16)
           (03 : DATA_TYPE_U32)
           (04 : DATA_TYPE_U64)
           (05 : DATA_TYPE_I8)
           (06 : DATA_TYPE_I16)
           (07 : DATA_TYPE_I32)
           (08 : DATA_TYPE_I64)
           (09 : DATA_TYPE_F32)
           (0A : DATA_TYPE_F64)

Item Type
    Data Type
        Data Len
                FMMU Index
|   |   |       |   |
 01  01  01  00  00 <- Item 0
 02  02  02  00  00 <- Item 1
 01  01  01  00  00 <- Item 2
 02  02  02  00  00 <- Item 3
 03  02  02  00  00 <- Item 4
 03  02  02  00  00 <- Item 5
```

SYBERA Copyright © 2014

# 8 EtherCAT Verifier (ECATVERIFY)

The EtherCAT Verifier Software is a powerful software to check and configure EtherCAT devices, without the need of programming. The Software guides interactively through all devices states and configuration steps and gives useful hints for programming. The Application ECATVERIFY is based on the Realtime EtherCAT Master Library and uses its exported functionality. To start its first required to init the realtime core and the ethernet transport layer. Therefore the NIC adapter (which is connected to the realtime core) has to be selected, as well as the sampling realtime period, the synchronisation cycles and eventually the scheduling count of the realtime application task (usually set to 1)



SYBERA Copyright © 2014

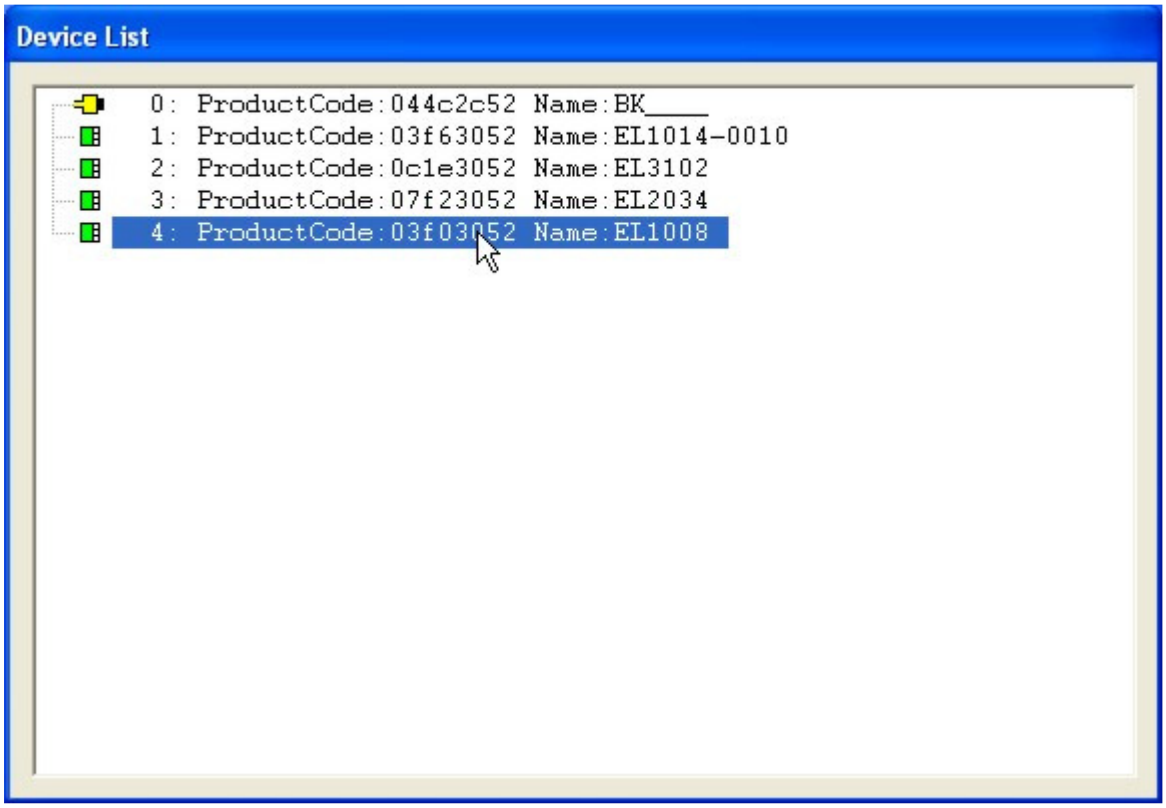
The initializing process is based on following library functions, defined in ECATCOREDEF.H and SHAECATCORE.H:

```
//Required ECAT parameters
ECAT_PARAMS EcatParams;
EcatParams.PhysAddr = DEFAULT_PHYSICAL_ADDRESS;      //0x10001000
EcatParams.LogicalAddr = DEFAULT_LOGICAL_ADDRESS;    //1000
EcatParams.SyncCycles = DEFAULT_SYNC_CYCLES;         //20
EcatParams.EthParams.dev_num = 0;
EcatParams.EthParams.eth_type = ETH_TYPE_ECAT;       //Set ethernet frame type
                                                     //filter for selected
                                                  interface
EcatParams.EthParams.eth_if = ETH_IF_CORE;           //Set filter interface
      (Send
                                                     //all other ethernet frames
                                                     //to socket interface)
EcatParams.EthParams.period = 100;                   //Set realtime period
      [µsec]
EcatParams.EthParams.sched_cnt = 1;                  //Set application task
                                                     //scheduler count (cycle
                                                  time =
                                                     //sched_cnt * period)
EcatParams.EthParams.fpAppTask = AppTask;
```

SYBERA Copyright © 2014

## 8.1 Device List

After the core has been initialized, the EtherCAT Master Library scans the bus for EtherCAT Slave Devices. A device list dialog appears from which devices may be selected for further processing. Devices can be selected by a “Left Mouse Double Click” on the corresponding line.



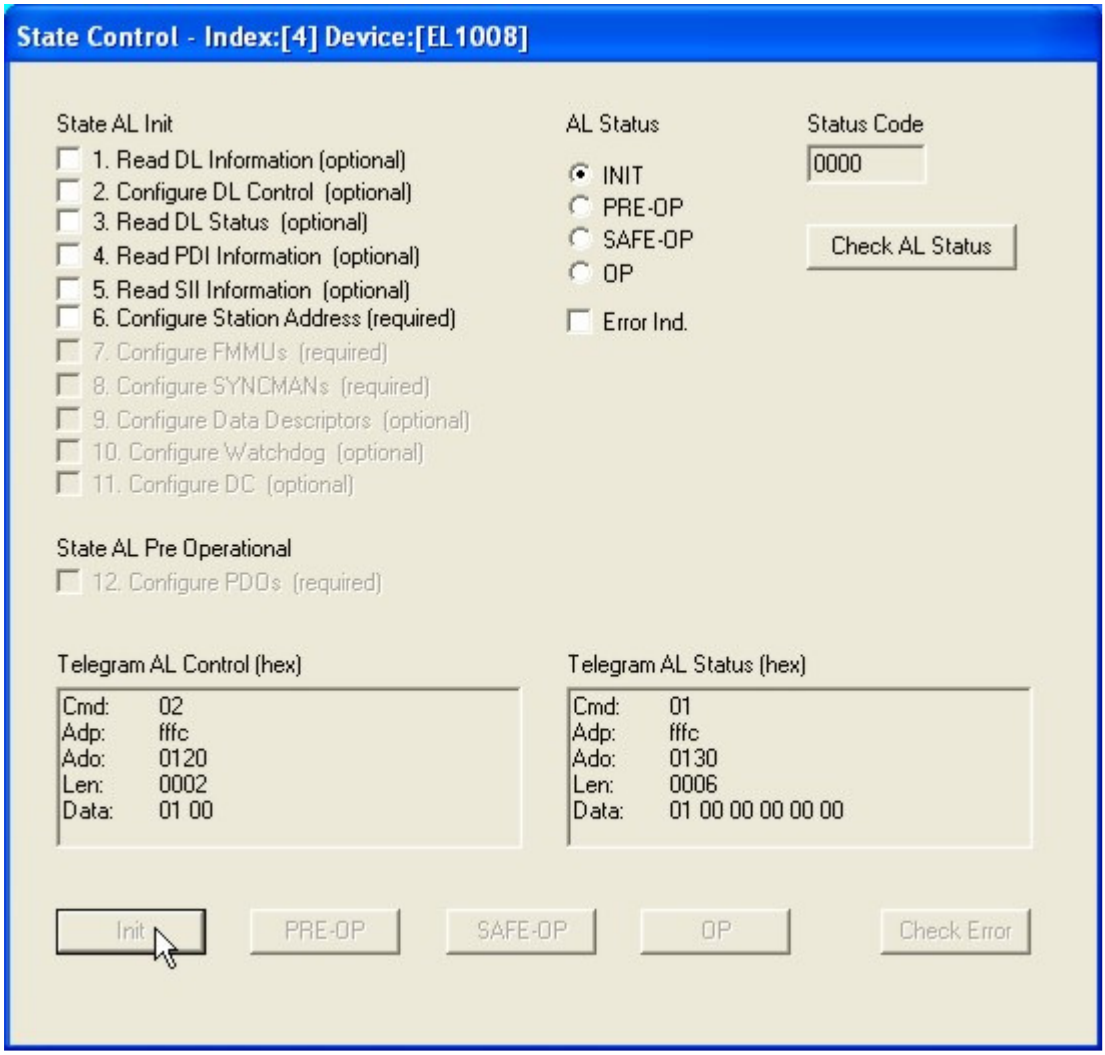

Note: With ECATVERIFY only the selected device will be enabled for further processing:

```
//Enable selected station
for (ULONG i=0; i<__StationNum; i++)
      if (i == m_StationIndex)      { __pUserList[i].bDisable = FALSE; }
      else                          { __pUserList[i].bDisable = TRUE; }
```

SYBERA Copyright © 2014

## 8.2 State Control Dialog

The state control dialog allows configuring the EtherCAT device with all required parameters and guide it step by step into the operating mode. Thereby some settings are required (like Station Address, FMMU, SYNCMAN and PDO), while other settings are optional (or only inforamtional. These settings are to be done by the corresponding configuration dialog. On each device State (INIT, PREOP, SAFEOP, OP) different setting are valid (due to the requirements of the EtherCAT specification). The State Control Dialog enables only these configuration abilities, which are currently valid, unless the required tasks have been fulfilled.




After pressing the INIT Button, the abilities 1 – 6 are enebled. Each configuration dialog contains additionally information about the corresponding EtherCAT telegram, which will be sent or received.

SYBERA Copyright © 2014

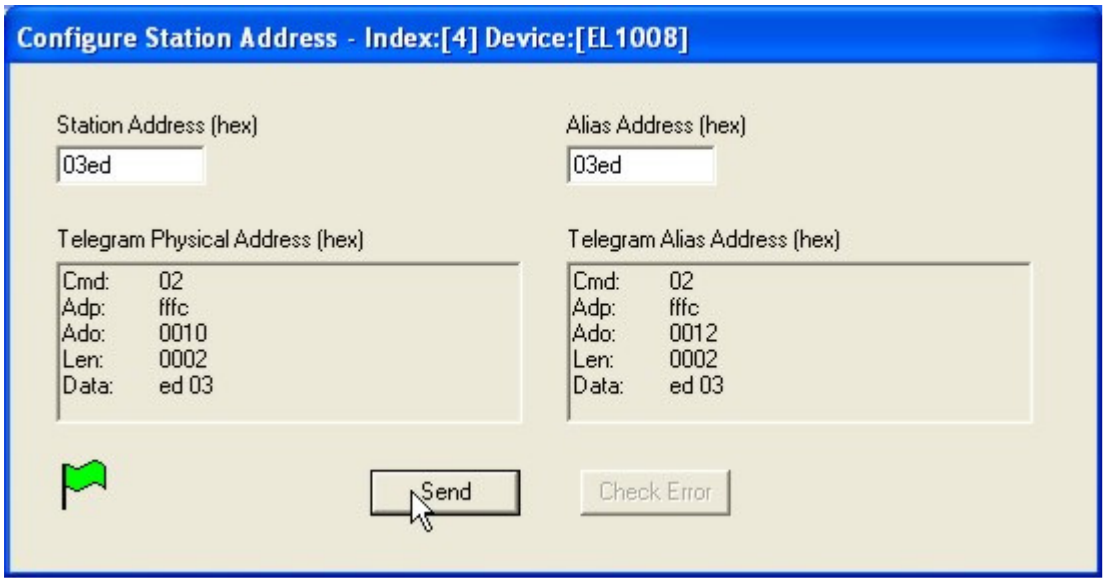
This configuration task uses the Library function(s), defined in ECATCOREDEF.H and SHAECATCORE.H:

```
//Define AL states
#define AL_STATE_INIT               0x01
#define AL_STATE_PRE_OP             0x02
#define AL_STATE_BOOTSTRAP          0x03
#define AL_STATE_SAFE_OP            0x04
#define AL_STATE_OP                 0x08


ULONG Result = = EcatChangeAllStates(AlState);
```

SYBERA Copyright © 2014

### 8.2.1 Configure Station Address

The station address must be configured by at least its physical address. Some newer devices allow configuring an additional ALIAS address



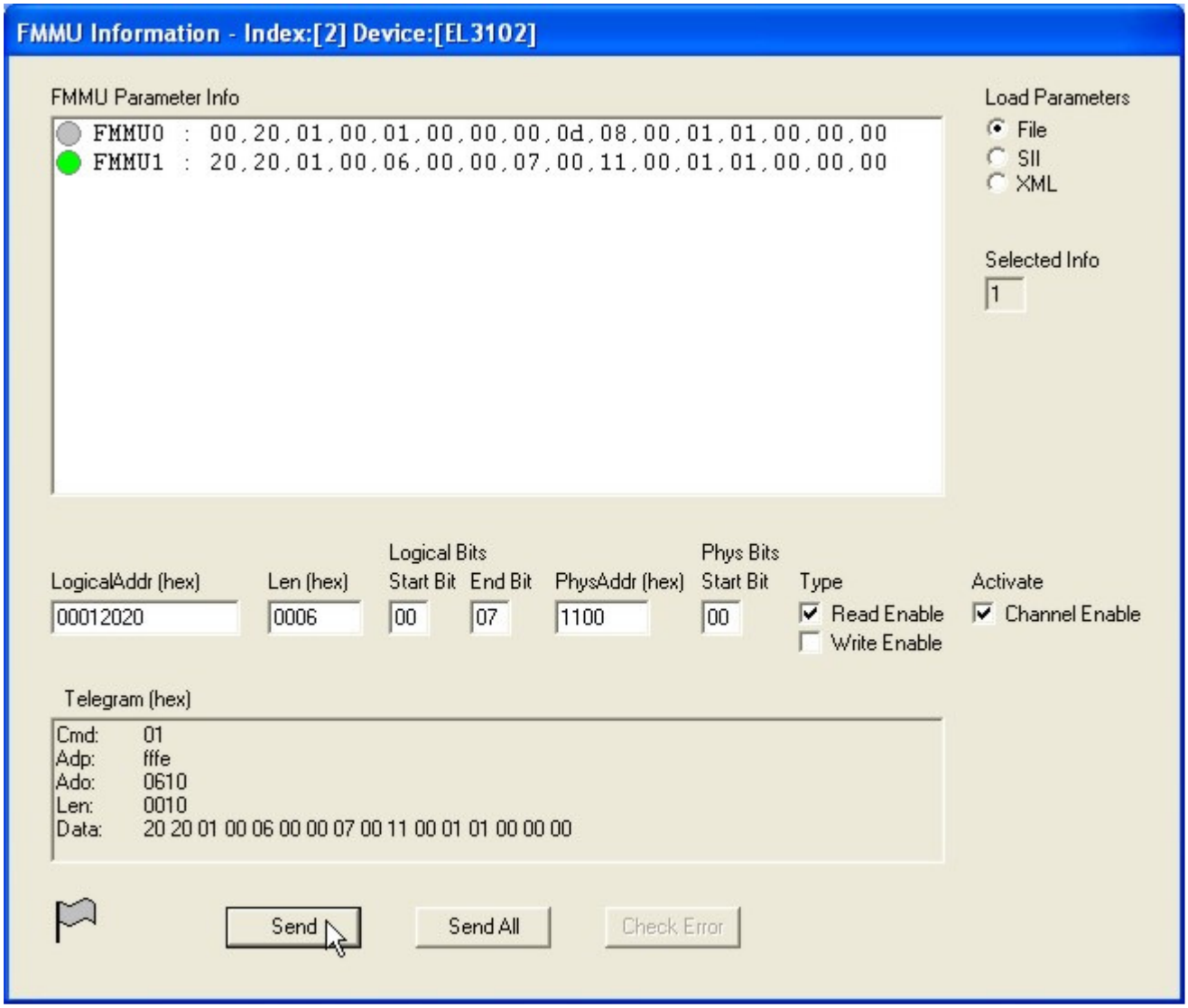

This configuration task uses the Library function(s), defined in ECATCOREDEF.H and SHAECATCORE.H:

```
ULONG Result = EcatInitStationAddresses(EcatParams.PhysAddr)
```

SYBERA Copyright © 2014

### 8.2.2 Configure FMMU Management

The FMMU ability dialog allows parsing XML information, EEPROM (SII) information and the Native format for configuration and provides information to all items (also described in the EtherCAT specification).



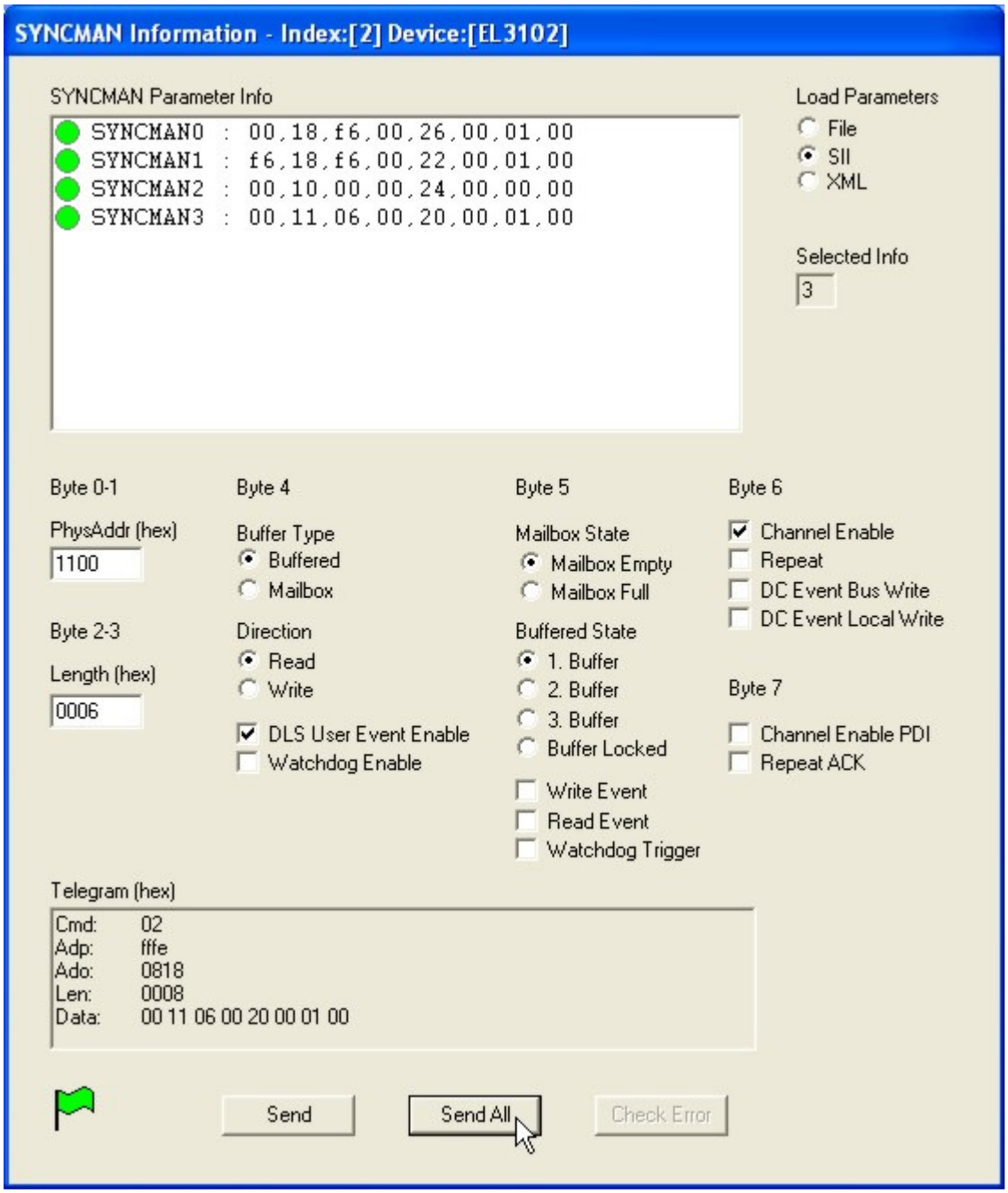

Each FMMU information can be selected by “Left Mouse Double Click” on the corresponding line. When the FMMU is selected it can be sent to the device. When all FMMU information is sent, configuration task is fulfilled. This configuration task uses the Library function(s), defined in ECATCOREDEF.H and SHAECATCORE.H:

```
ULONG Result = EcatInitFmmus(EcatParams.LogicalAddr);
```

SYBERA Copyright © 2014

### 8.2.3 Configure SYNC Management

The SYNCMAN configuration dialog allows parsing XML information, EEPROM (SII) information and the Native format for configuration and provides information to all items (also described in the EtherCAT specification).



SYBERA Copyright © 2014

Each SYNCMAN information can be selected by “Left Mouse Double Click” on the corresponding line. When the SYNCMAN is selected it can be sent to the device. When all SYNCMAN information is sent, configuration task is fulfilled. This configuration task uses the Library function(s), defined in ECATCOREDEF.H and SHAECATCORE.H:

```
ULONG Result = EcatInitSyncManagers();
```

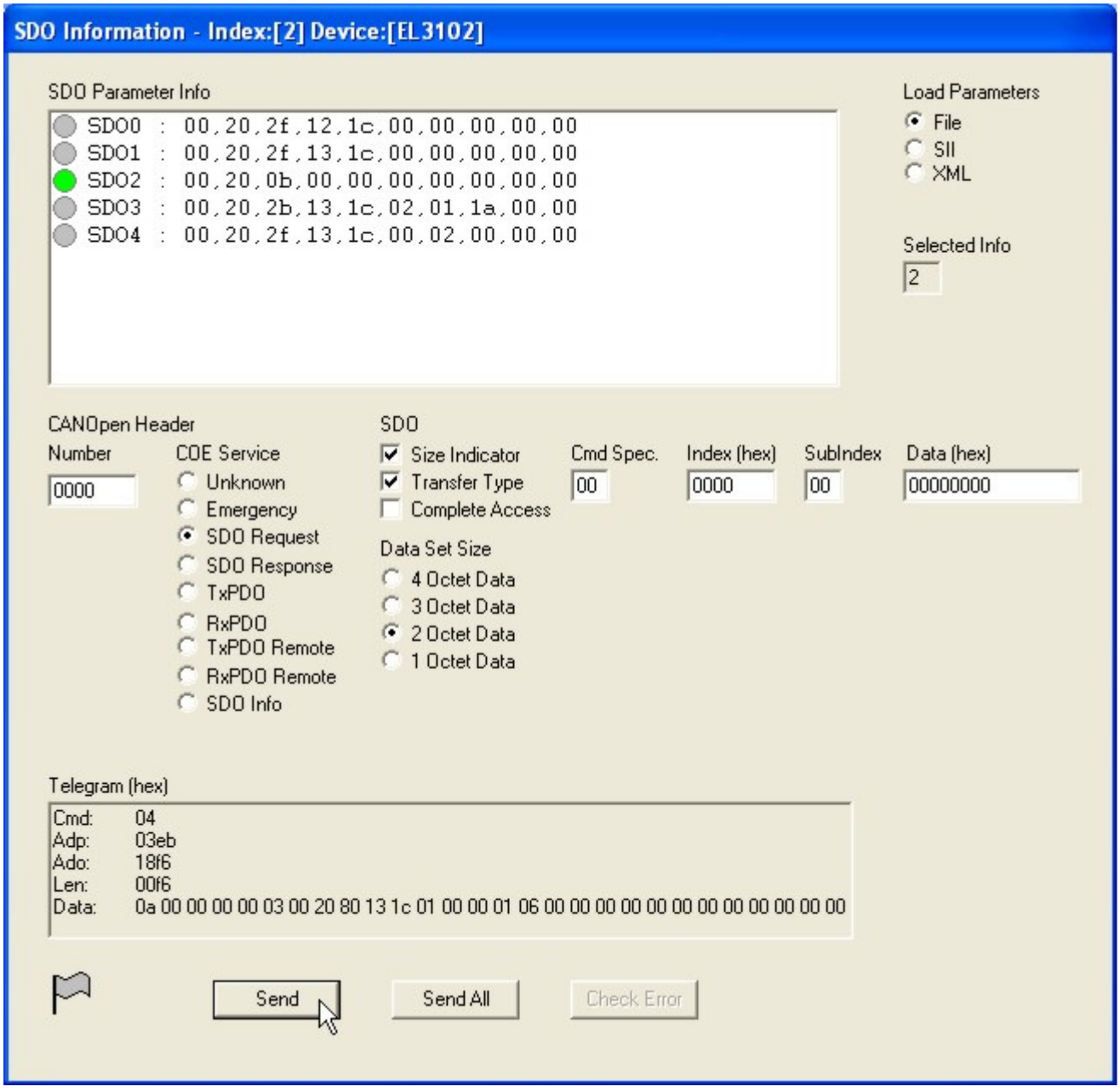
Note: After configuring the SYNC Managers all required configuration tasks within the INIT State are fulfilled. The next state PREOP is now required:

```
ULONG Result = EcatChangeAllStates(AL_STATE_PRE_OP);
```

SYBERA Copyright © 2014

### 8.2.4 Configure PDO(s)

The PDOs (Process Data Objects) are typically sent by COE (Can Over Ethernet) with use of mailbox communication. The COE Mailbox communication uses SDOs (Service data Objects) to provide the PDO information to the device. Thus the native format describes SDOs instead of PDO data. The PDO (SDO) configuration dialog allows parsing XML information, EEPROM (SII) information and the Native format for configuration and provides information to all items (also described in the EtherCAT specification).



SYBERA Copyright © 2014

Each SDO information can be selected by “Left Mouse Double Click” on the corresponding line. When the SDO is selected it can be sent to the device. When all SDO information is sent, configuration task is fulfilled. This configuration task uses the Library function(s), defined in ECATCOREDEF.H, ECATSDODEF.H and SHAECATCORE.H:

```
ULONG Result = EcatPdoAssignment ();
```

Note: After configuring the PDO Assignment all required configuration tasks within the PREOP State are fulfilled. The next states SAFEOP and OP are now required:

```
ULONG Result = EcatChangeAllStates(AL_STATE_PRE_OP);
ULONG Result = EcatChangeAllStates(AL_STATE_OP);
```

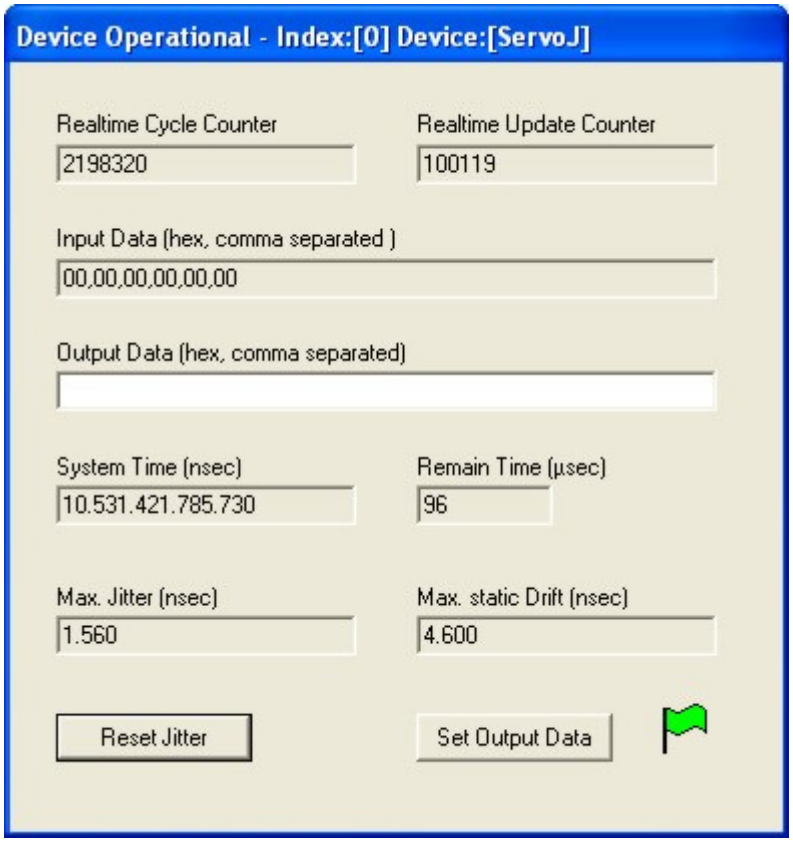
SYBERA Copyright © 2014

### 8.2.5 Device Operational

When changing the state to operational, device is updated by realtime cycles. Each update cycle sets and gets the station telegrams TxTel and RxTel:

```
__pSystemList[StationIndex].TxTel.s.data[DataOffset] = OutputValue;
InputValue = __pSystemList[StationIndex].RxTel.s.data[DataOffset];
```

Since many devices support Distributed Clock management, the local system time of the device allows exact jitter and drift measurement.

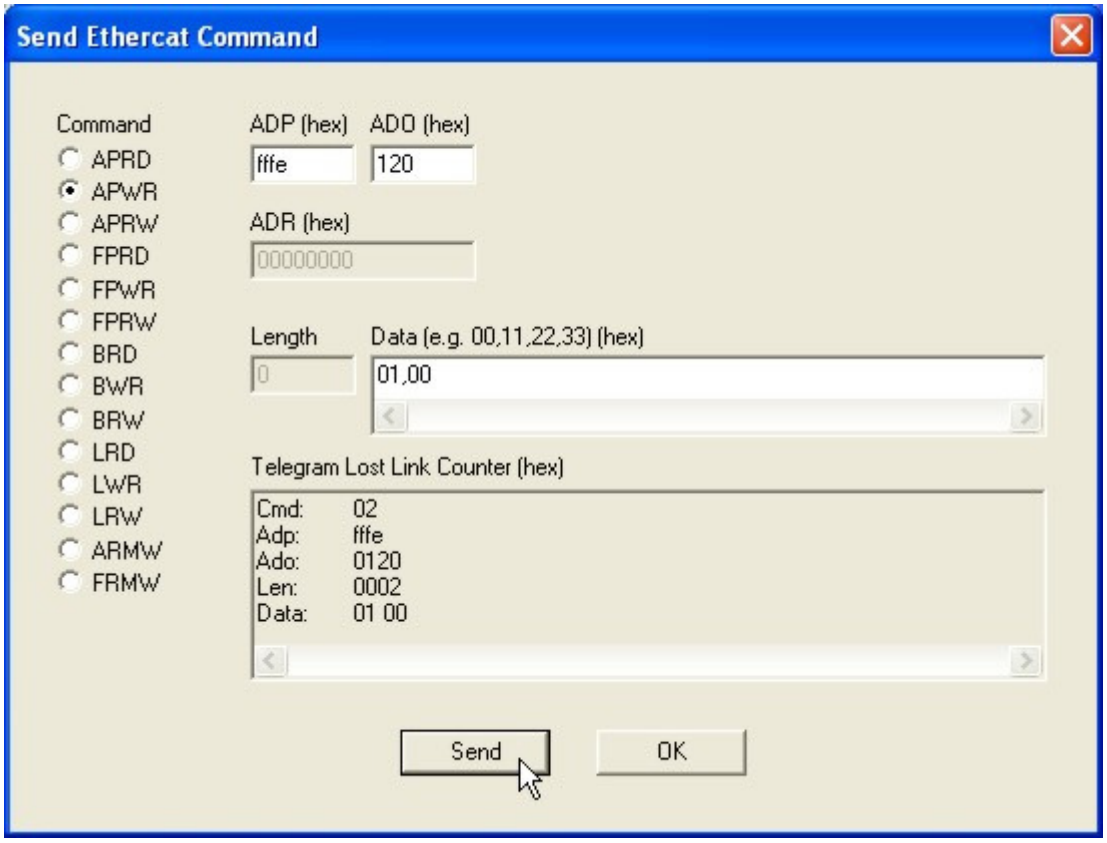



Note: Not each sampling cycle updates the device, since the realtime cycle is typically much faster than the synchronisation cycle. This is why the realtime cycle counter differs to the update counter.

SYBERA Copyright © 2014

## 8.3 Sending EhterCAT Command

ECATVERIFY allows building and sending of single EtherCAT Commands for test purposes.

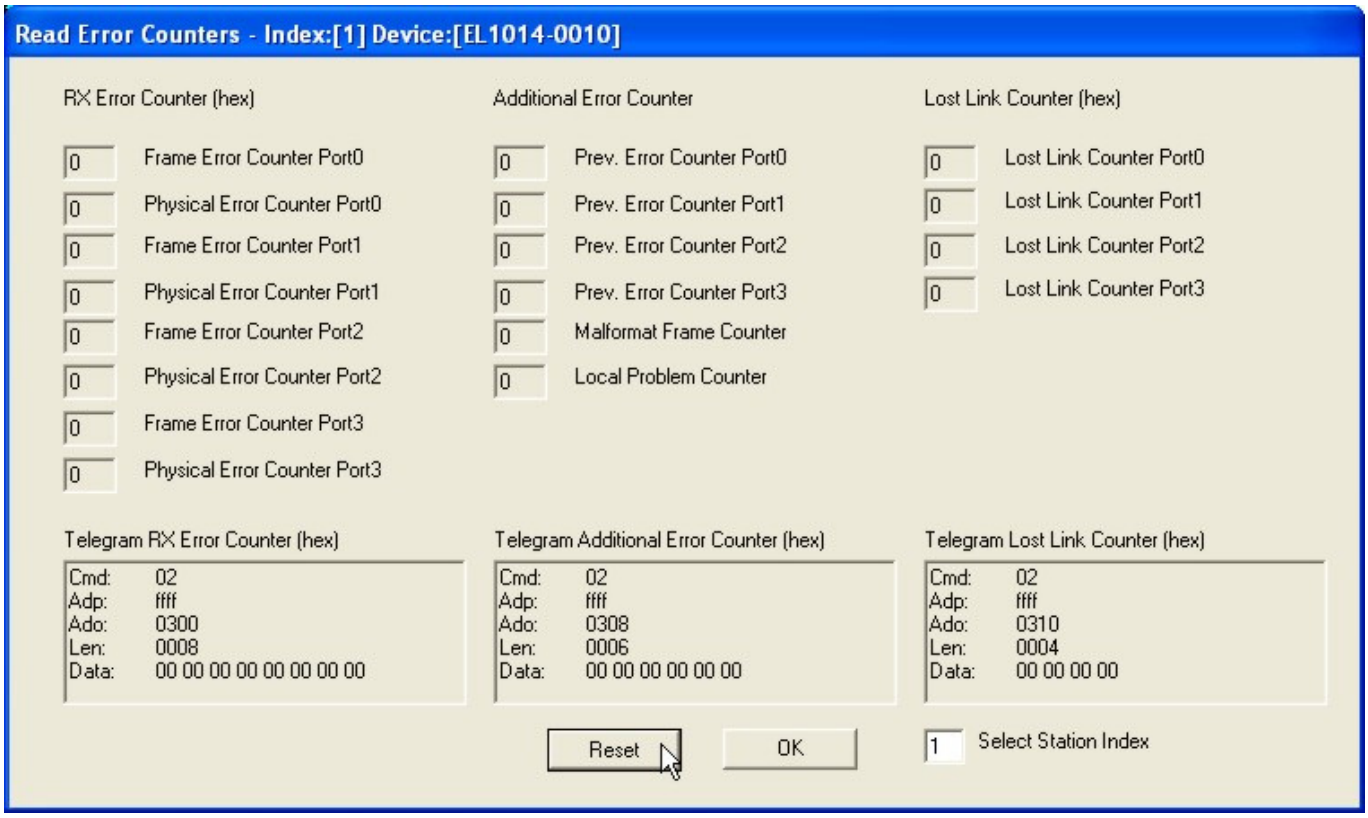


SYBERA Copyright © 2014

## 8.4 Error Counters

ECATVERIFY gets information about the ErrorCounters

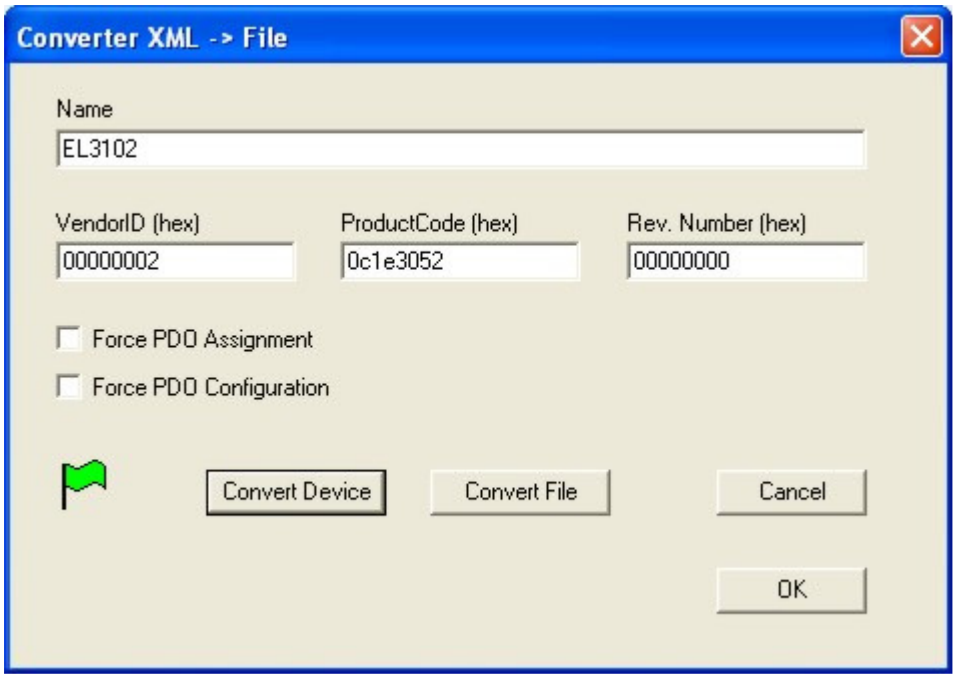
- RX Error Counter
- Additional Error Counter (if supported by the device)
- Lost Link Counter (if supported by the device)



SYBERA Copyright © 2014

## 8.5 XML Converter

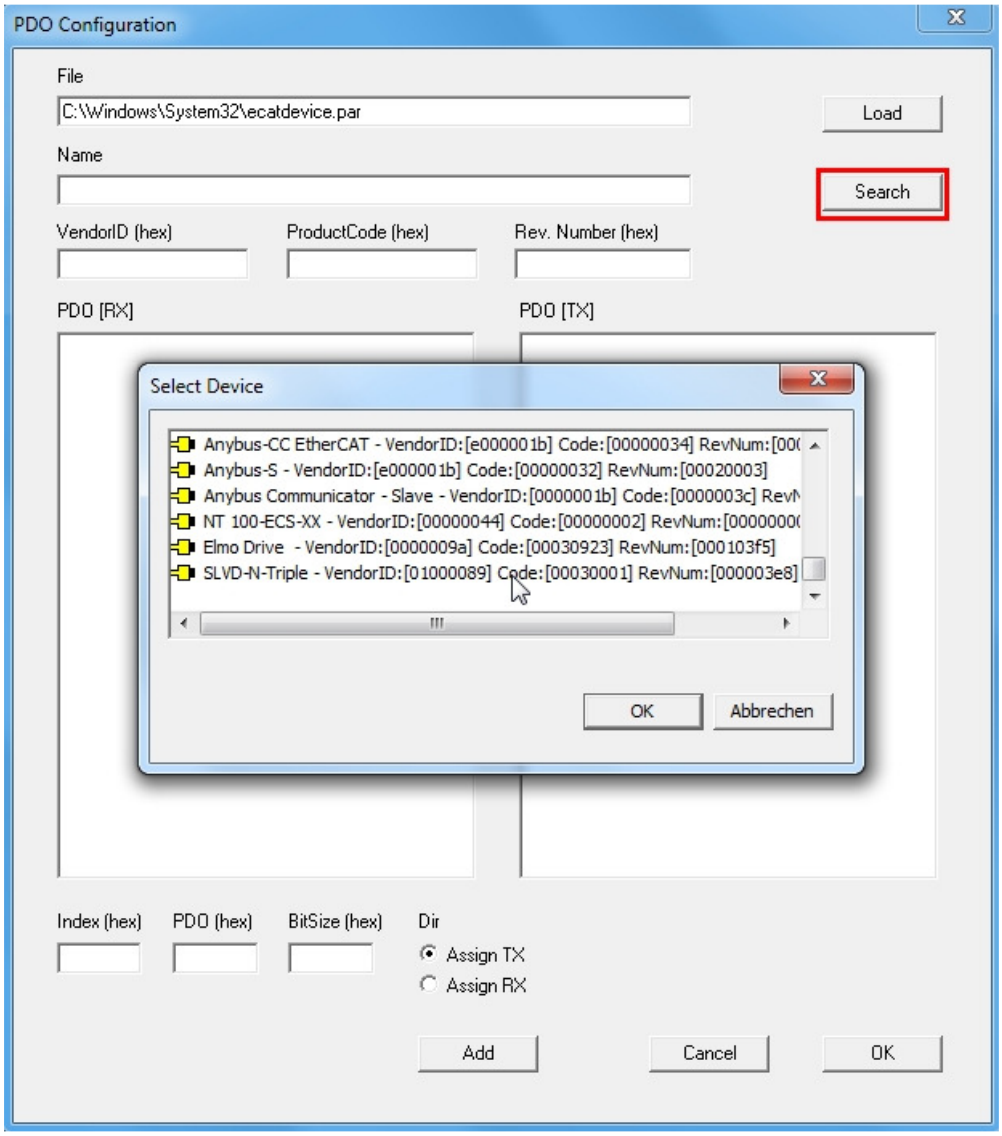
ECATVERIFY has an implemented XML parser which allows converting XML (ESI) device information into Native Parameter and save it into the parameter file ECATDEVICE.PAR (to be placed in \WINDOWS\SYSTEM32). Therefor the XML files must be located in the directory where ECATVERIFY is located. The device which is to be converted may be searched within an XML file by its Name, Product Code, Vendor ID or Revision Number. Its also possible to convert the whole XML file to the native format. Devices which are already present in ECATDEVICE.PAR will be updated.



SYBERA Copyright © 2014

## 8.6 PDO Configurator

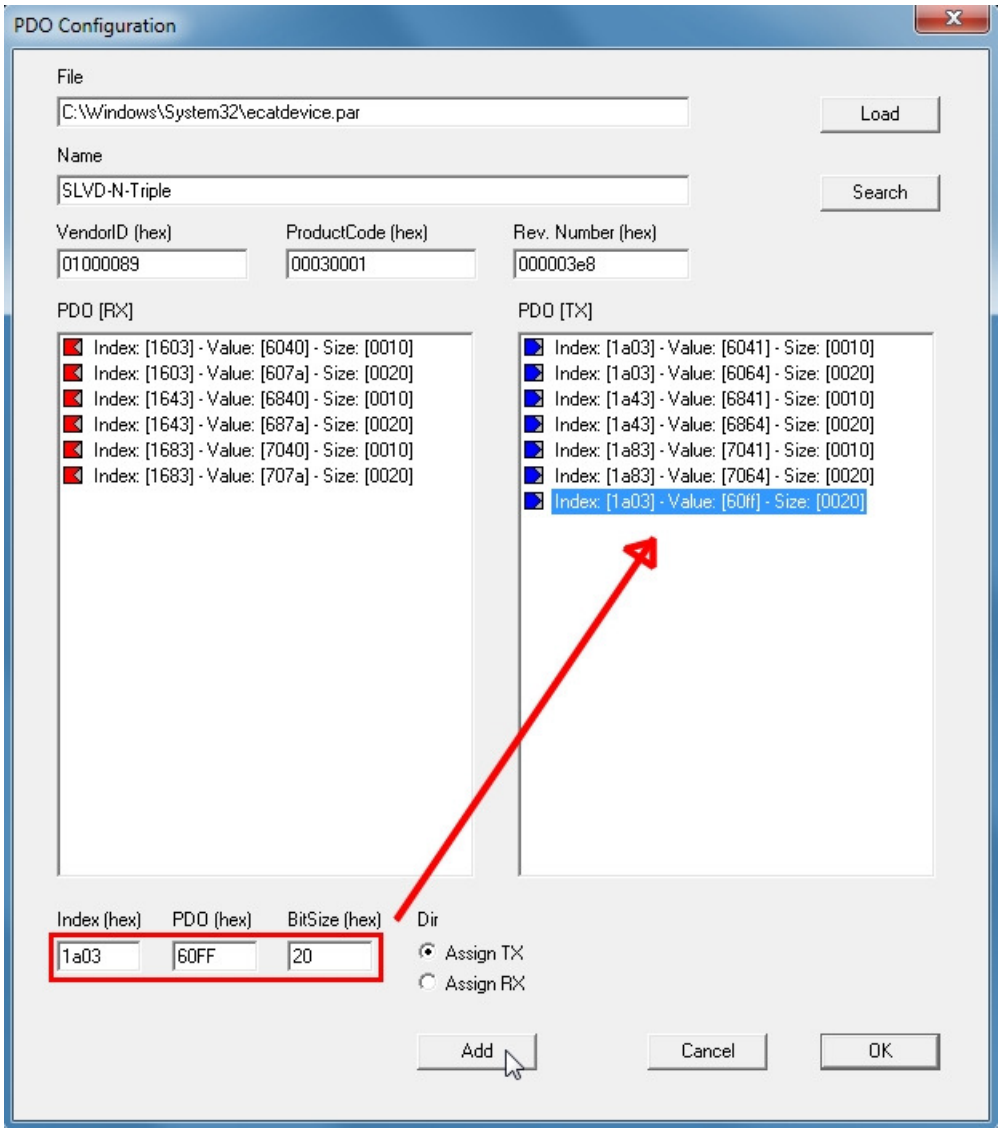
The integrated PDO configurator allows easy determination of the EtherCAT PDO mapping. The PDO Configurator allows adding, removing, and deleting PDO mapping objects. With the PDO-Configurator devices located in the file ECATDEVICE.PAR can be listed or searched for editing the PDO mappings.




Note: Existing PDO-Mappings need to have an already listed PDO assignment (1C12 / 1C13). Otherwise the PDO mapping has to setup newly.

SYBERA Copyright © 2014

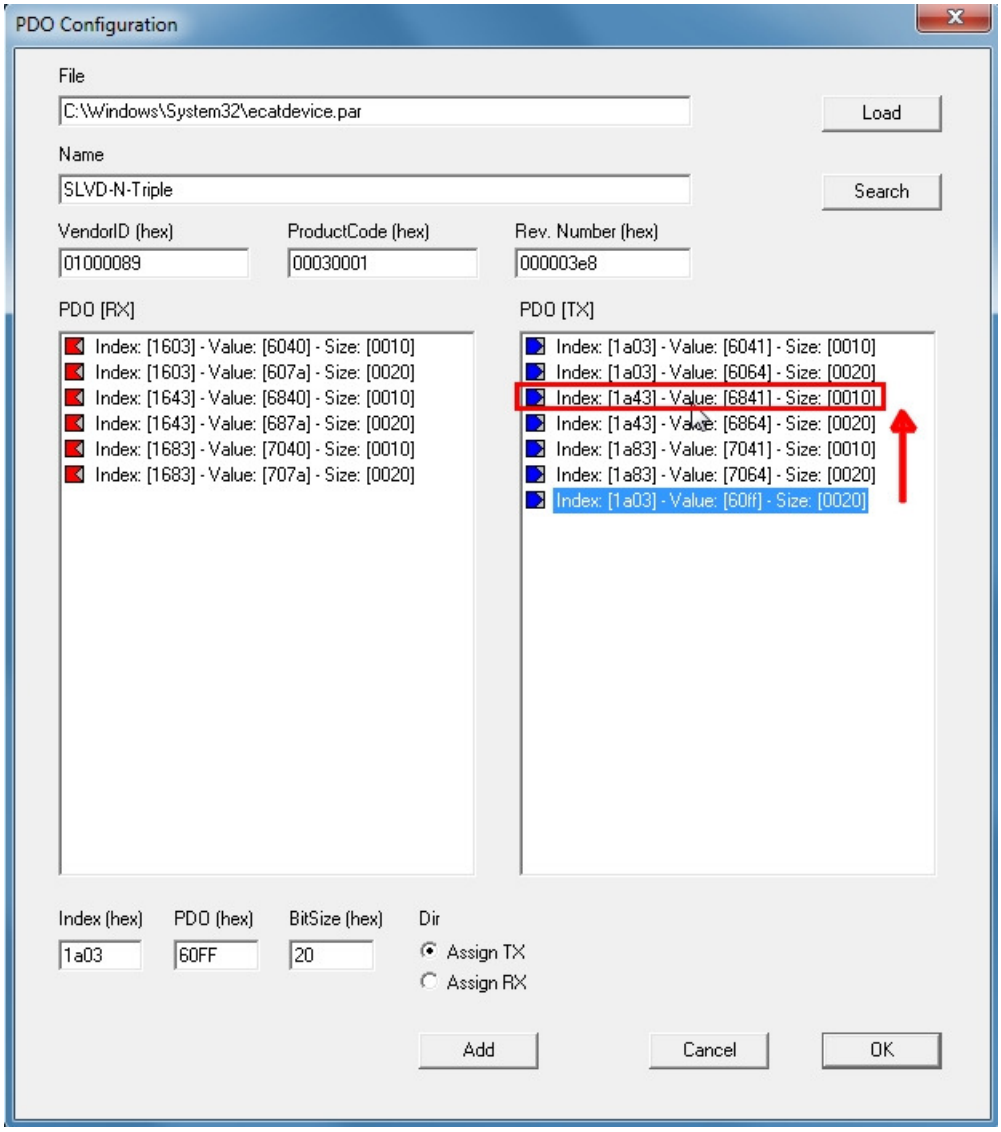
New PDO mappings are entered by index, PDO and bit size for assigning it to the corresponding PDO mapping list (TX / RX).




Selected PDO mappings may be deleted by pressing the key „DELETE“.

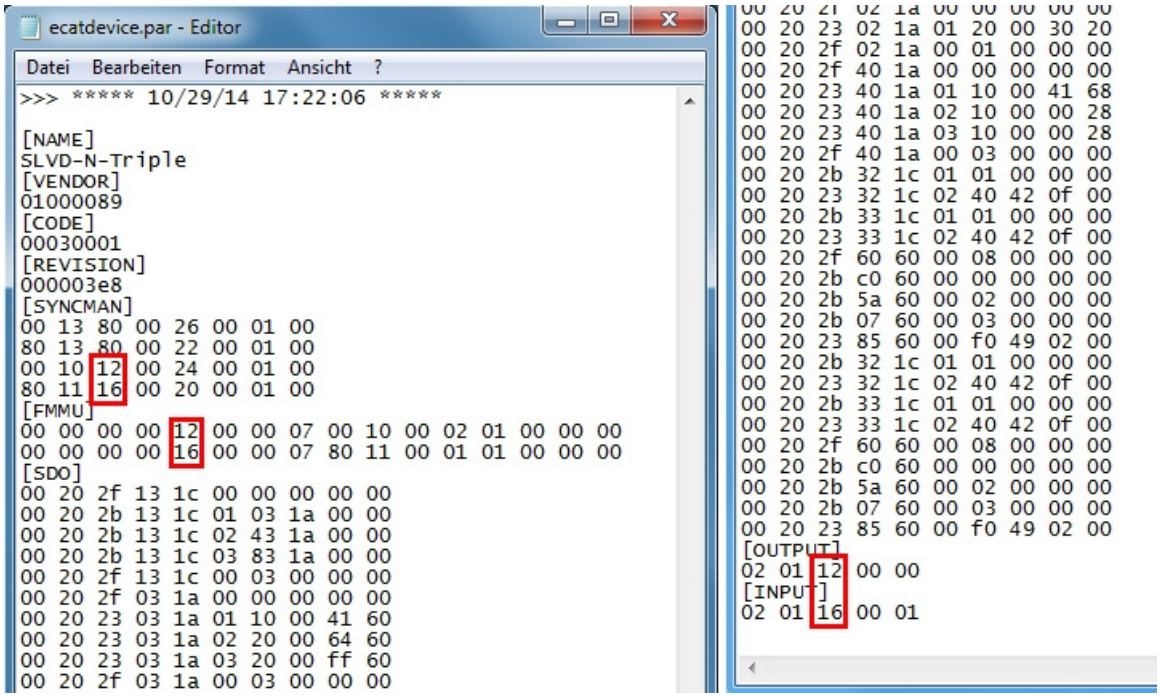
SYBERA Copyright © 2014

The new PDO mapping entries can be moved to the appropriate position. For this, the corresponding entry is selected to be moved and swapped with the entry of the desired position by clicking on it.



SYBERA Copyright © 2014

Once configured, the device located in the file ECATDEVICE.PAR file is automatically updated and the value “length” of the corresponding FMMU-, SYNCMAN- and INPUT / OUTPUT descriptor entries is automatically updated.



SYBERA Copyright © 2014

# 9 Error Handling

The master library provides an error handling and tracing mechanism.

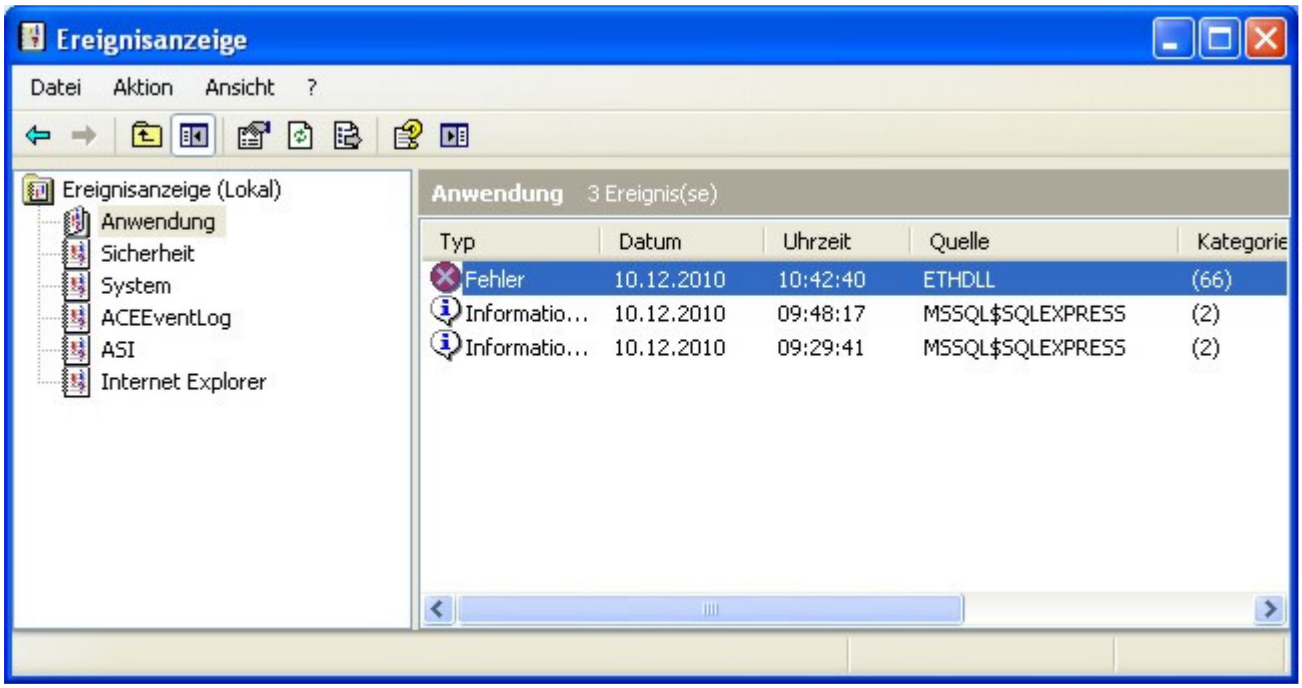
## 9.1 Debug LOG File

On execution the master library creates a sequence file ECATDBG.LOG in Text-Format

Note: This file is not accessible while the application is running

## 9.2 Event File

On execution the master library logs error event to the Windows Event Manager. The master library logs Application and System events. These events can be exported to a file and provided for support purposes.



SYBERA Copyright © 2014

# 10 Related Dokuments

- manual_sha_e.pdf (SHA Realtime Library)
- manual_eth_e.pdf (ETH Realtime Library)