EtherCAT Realtime Master Library Documentation (Cluster 64 Bit)

Date: July, 26.2022

EtherCAT Realtime Master Library Documentation

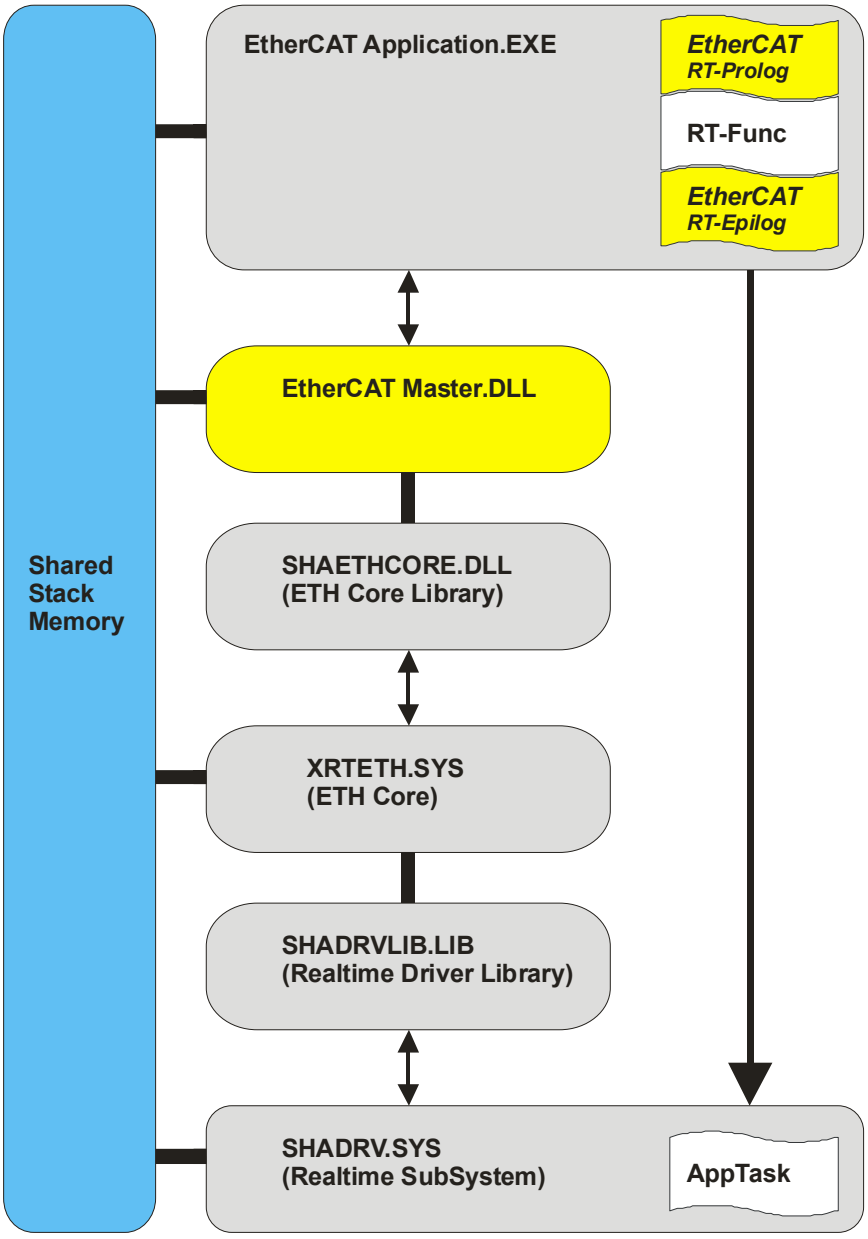
1 Introduction

With the PC-based EtherCAT Master Stack for Windows and the X-Realtime Engine the need for a separate controller hardware has been eliminated, as the Master Control is implemented directly from PC with standard Ethernet adapters. With the X-Realtime Engine of SYBERA a standard Ethernet card can be used as EtherCAT Master directly. The physical connection is made via a standard INTEL or REALTEK PCI or a corresponding PCMCIA or PCIe adapter. Basis for this is the SYBERA Master protocol stack and the X-Realtime Technology. The software runs ander Windows, and allows the control of slave devices (e.g. the EtherCAT modules of Beckhoff GmbH) in realtime. Depending on the PC hardware and application telegram update times are feasible upto 50 micro-sec. The latest version of the EtherCAT master controls Drives from Beckhoff, LTi, Metronix, Kollmorgen, ELMO (and more) DC synchronized with a sampling rate of 100 microseconds and an update period of 1 msec (changeable). For controlling Drive controllers SYBERA uses the procedure "Dynamic Jitter Compensation" with active and passive feedback. The PDO parameters are set with the EtherCAT Configurator.

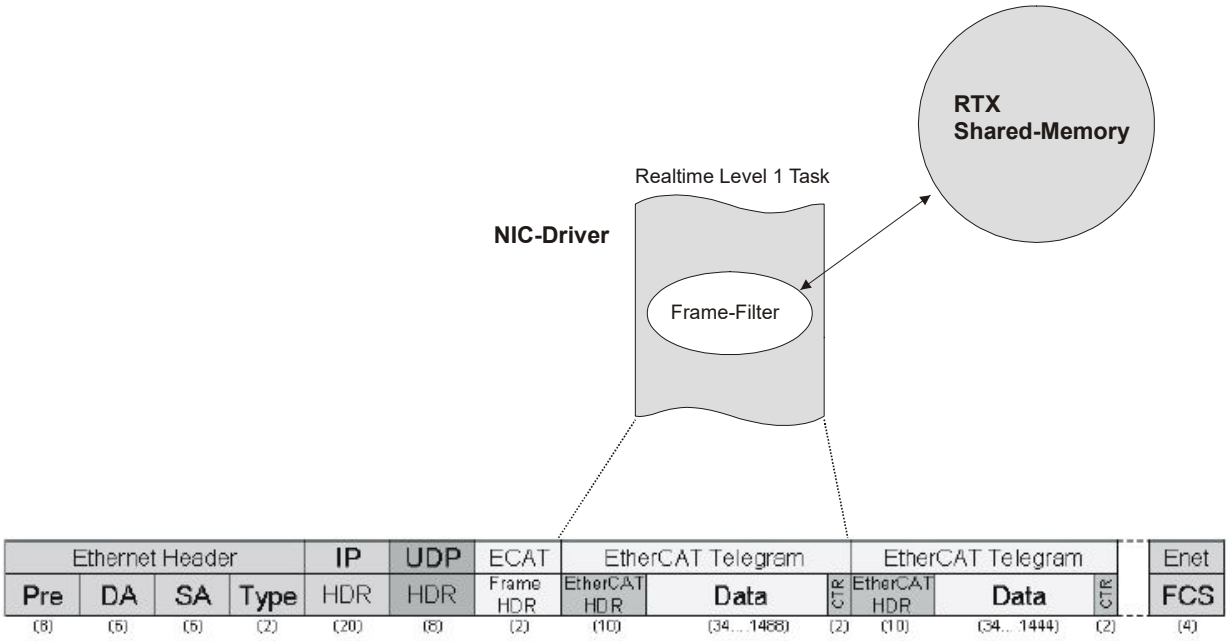
Beside numerous extended functions for Distributed Clock, COE and State management, the library system also allows to control EtherCAT devices, even without a corresponding XML file. With the integrated station management the devices may be completely administered and controlled almost implicitly, or 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. All information can be visualized in detail here. The integrated PDO Configurator allows easy definition of the PDO mappings for EtherCAT devices. With the Configurator, adding, removing and moving of PDO objects is made easy. In the ECATDEVICE.PAR file registered devices can be listed or searched for processing the PDO mapping. New PDO mappings are entered with index, PDO and bit size, and the corresponding PDO mapping list (TX / RX) will be assigned. After Configuration the entry within file ECATDEVICE.PAR is automatically updated and the corresponding length of the FMMU-, SYNCMAN- and descriptors entries are restated.

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.



On this occasion, not only the sending and receiving of ethernet frames ander 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, and functional control. With an integrated state machine the tasks are functionally synchronized.
Internally it 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. In this case hhe developer has the opportunity, to implement the functional processing (Realtime Level2) in a real-time task on system or application level. SYBERA uses with the EtherCAT Master Library the procedure "Dynamic Jitter Compensation" with active and passive feedback. Although the X-Realtime Engine runs with a low jitter (depending on the hardware platform), it results in an additive jitter due to the systemic of sampling operation. The integrated Distributed Clock (DC) management of the Master Library compensates the time offset, the static and dynamic drift, as well as the propagation delay. Depending on the configuration of the drive the parameters (eg control word, status word, target position, current position) can be accessed directly in real time.
For the initialization of the drive the corresponding XML file is read and evaluated by the master and converted into corresponding commands.



1.1 Product Features

- Intelligent Station Management
- Station Realtime Cycles upto 100 µsec
- Logical, Physical and Alias Station Addressing
- Mailbox Interface and COE Management
- EOE Management
- FMMU Management
- SYNC Management
- PDO Assignment and PDO Configuration
- Distributed Clock Support and DC Configuration
- Watchdog Support
- State Management
- Combined Logic Support
- ESI Configurator
- Code Assistant
- EtherCAT Service

1.2 Supported Platforms

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

- Visual C++ (from Version 2010)
- CVI LabWindows
- Borland C++Builder

1.3 Supported OS

-
- Windows 7, 8, 10

2 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)
2. Make shure, the latest OS-Updates are installed
3. First install SHA realtime system (separate software library package)
4. Next install ETH transport library (separate software library package)

Installation

5. Run SYSETUP64.EXE 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.
   For the evaluation version use:

   PID: 11223344
   SERNUM: 11223344
   KEYCODE: 00001111-22223333

6. Optional: Check license with SYLICENCECHECK64.EXE

Operation

7. Run ECATVERIFY64.EXE
8. Build device description ECATDEVICE[name].PAR with the ESI configurator
9. Do the PDO Assignment with the PDO configurator
10. Bring up the device to operational with EcatVerify
11. Build the program with the library interface
12. Run the program

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

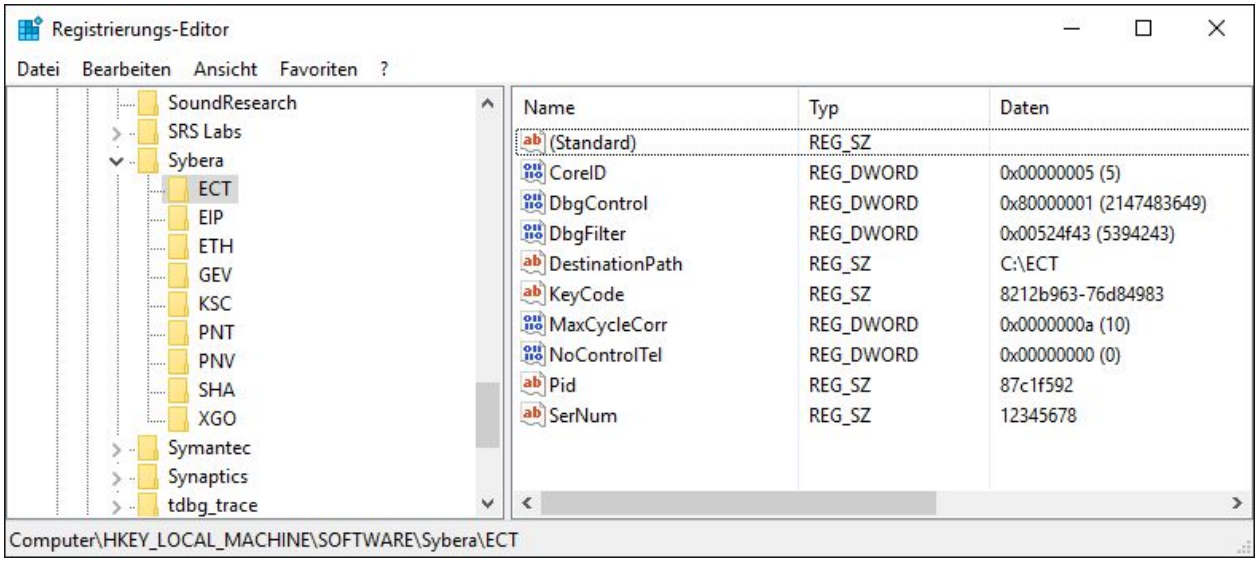
2.1 No adminstartor rights required

After installation, no administrator rights are required to run the master stack.

2.2 Registry Settings (optional)

SYBERA uses a PWM (pulse wide modulation) for the guidance of the master clock. Therefore following registry settings can be adjusted:

HKEY_LOCAL_MACHINE\SOFTWARE\Sybera\ECT



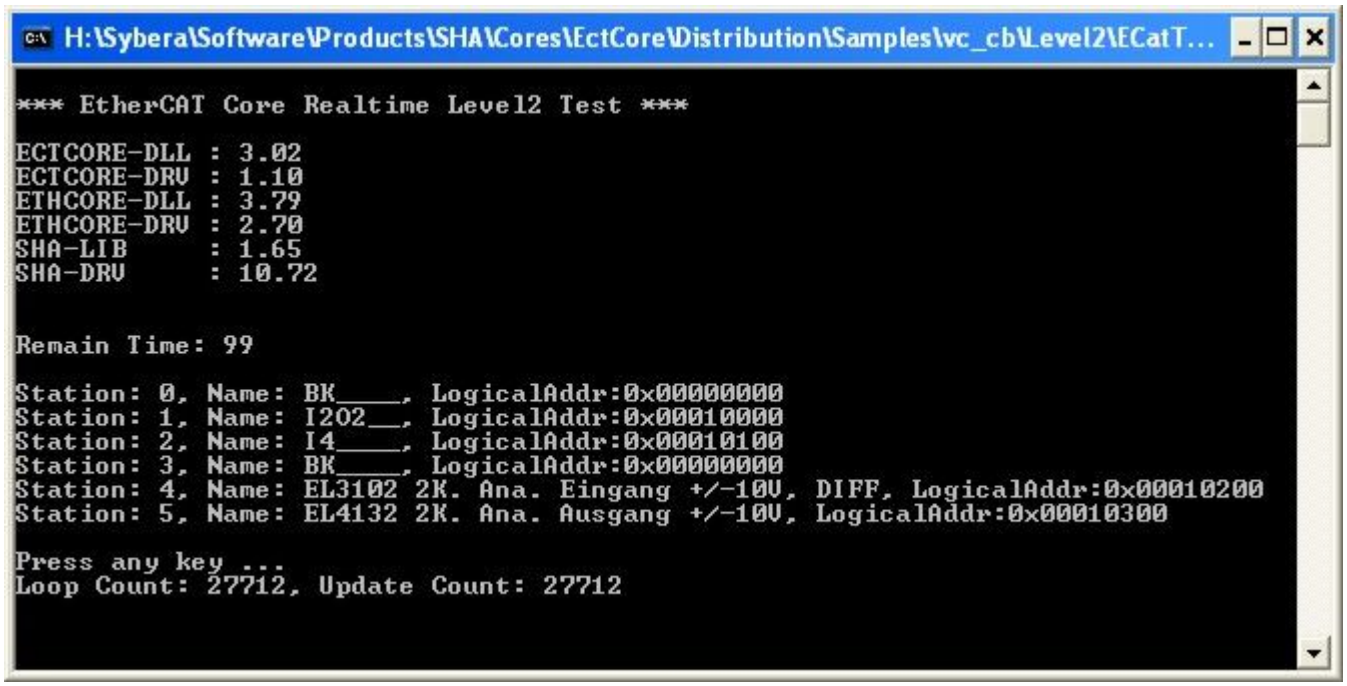

```
MaxCycleCorr = 0x0000000a
NoControlTel = 0x00000000
```

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 available:

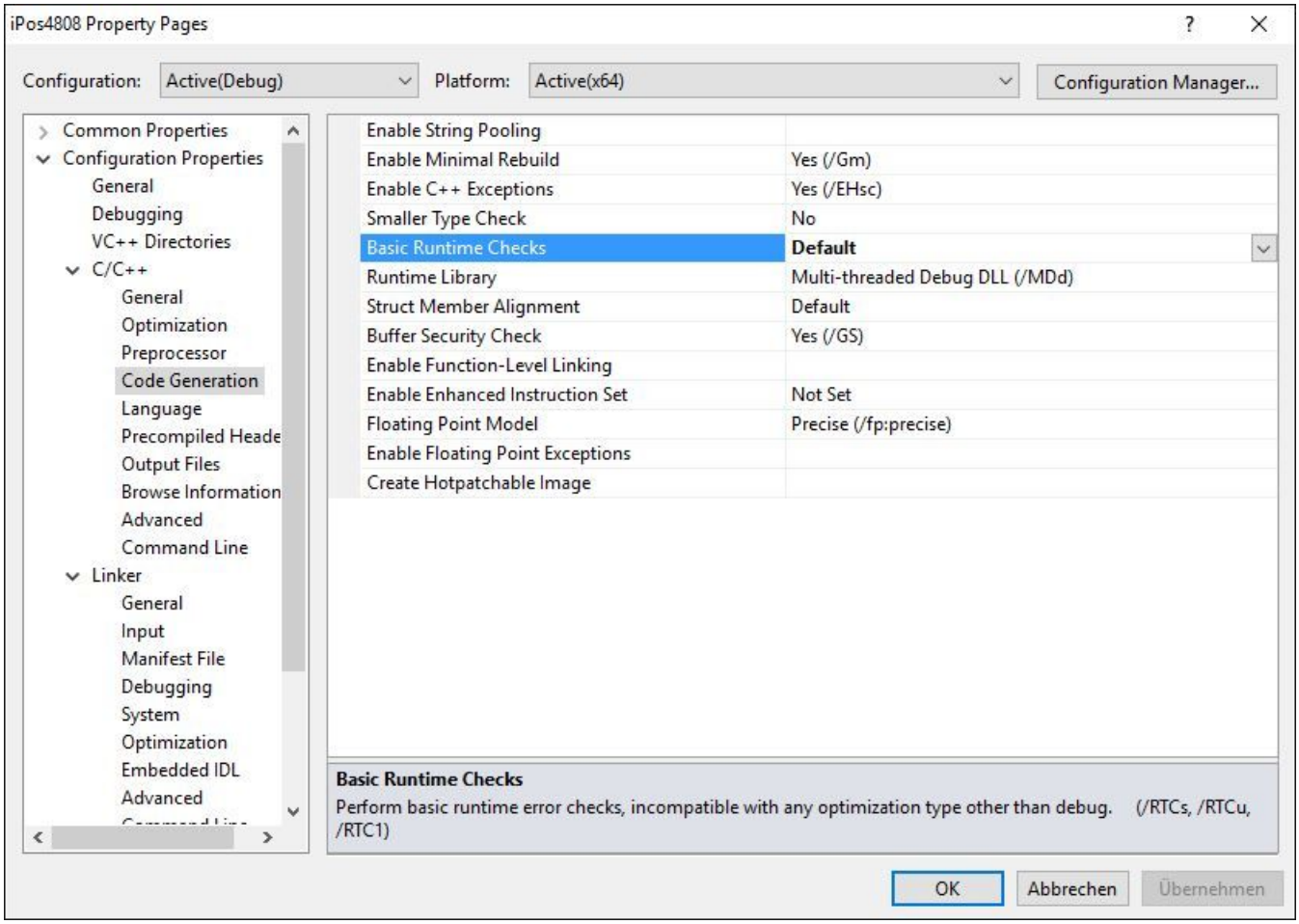
SHA*64*ECATCORE.DLL	EtherCAT Master DLL (VISUAL C++)
SHA*64*ECATCORE.LIB	EtherCAT Master LIB (VISUAL C++)
ECATDEVICE_[name].PAR	Native Station Configuration File
SHA*64*ECATCORE.H	Exported Function Prototypes
ECAT*64*COREDEF.H	EtherCAT Basic Definitions
ECAT*64*DCDEF.H	EtherCAT Distributed Clock Definitions
ECAT*64*EOEDEF.H	EtherCAT EOE Definitions
ECAT*64*MACROS.H	EtherCAT Macros
ECAT*64*CONTROL..H	EtherCAT Ethernet Management
ECAT*64*MAILBOXDEF.H	EtherCAT Mailbox Definitions
ECAT*64*REGDEF.H	EtherCAT Register Definitions
ECAT*64*SDODEF.H	EtherCAT COE Definitions
ECAT*64*SIIDEF.H	EtherCAT EEPROM Definitions
ECAT*64*SILENTDEF.H	EtherCAT Silent State Management

Sample Application



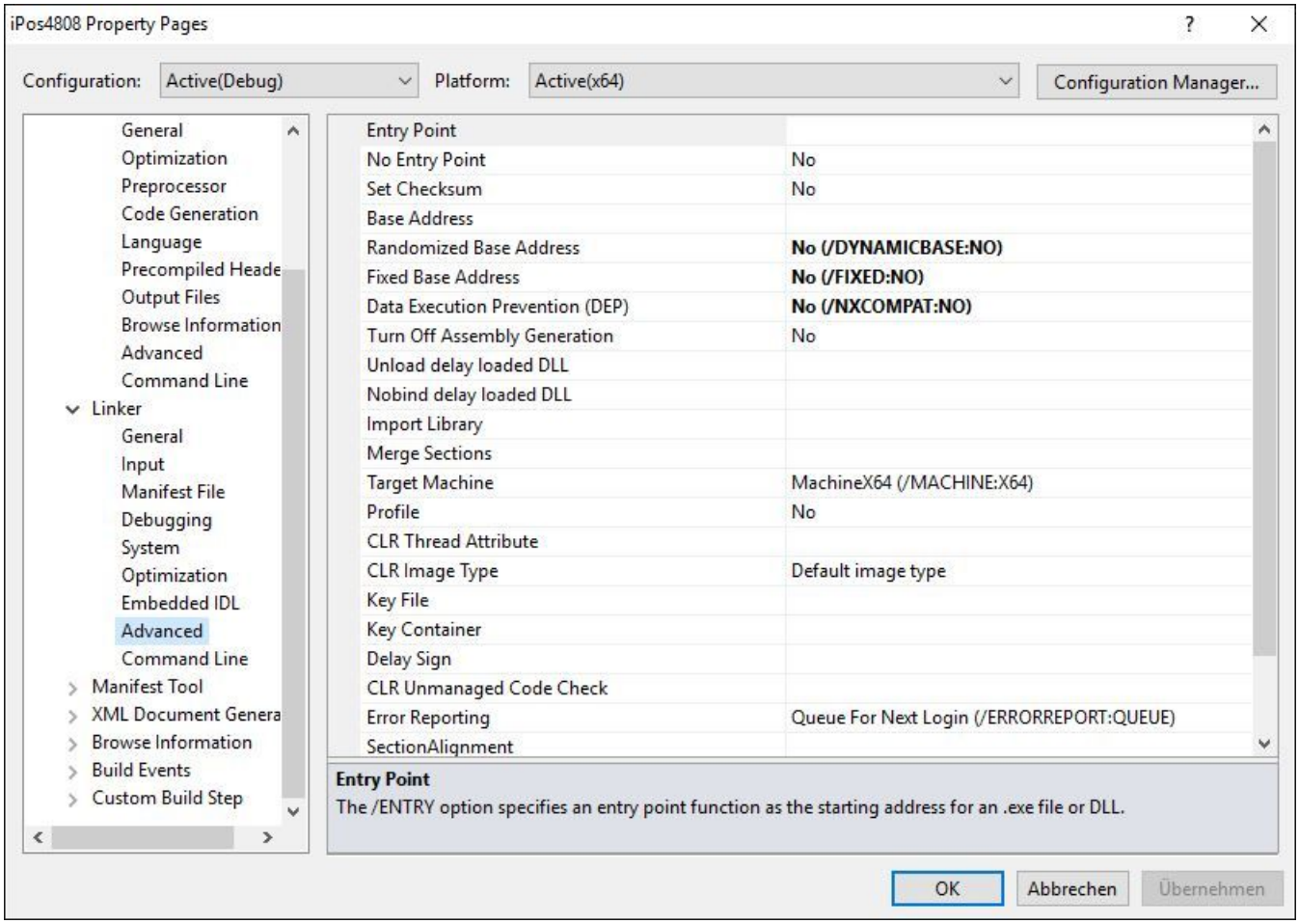
3.1.1 Visual Studio Compiler Settings

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



3.1.2 Visual Studio Linker Settings

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



3.2 Header File ECAT64COREDEF.H

The header file ECAT64COREDEF.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 Beginner 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.

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
     EXT_STATION_INFO   ExtInfo;                //Extended station info
     UCHAR              Reserved[RESERVED_SIZE];     //Reserved Data Size

} STATION_INFO, *PSTATION_INFO;
```

Note:

- The EtherCAT structures are described in detail inside the EtherCAT specification and are only used for the development with the EtherCAT Library Expert Interface.
- Since most Library Expert 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 RESERVED_SIZE
- For accessing the realtime process telegrams TxTel and RxTel use the macros defined in ECAT64MACROS.H

3.2.3 Structure EXT_STATION_INFO

This structure extends the information of STATION_INFO and is for internal use only.

```
typedef struct _EXT_STATION_INFO
{
      LONG  ShiftTimeSync0;         //SYNC0 Shift Time
      LONG  ShiftTimeSync1;         //SYNC1 Shift Time
      SHORT PortCnt;                //Port Count
      ULONG DiffDelay;              //Difference Delay
      ULONG PropDelay;              //Propagation Delay
      SHORT MailboxCnt;             //Mailbox Counter
      ULONG LogicalDataOffs[FMMU_MAX_DIR];     //Logical data offset

} EXT_STATION_INFO, *PEXT_STATION_INFO;
```

3.2.4 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, data len and assigned FMMU index.

```
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  (bytes)
    UCHAR    Fmmu;  //Assigned FMMU index

} DATA_DESC, *PDATA_DESC;
```

The data descriptors may be used to initialize process telegrams with the Library Expert Interface (it's not required when using the Library Beginner interface function ShaEcatEnable):

(Sample FMMUCONTROL)

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

     //Get logical address of first descriptor, if descriptors are present
     if (pStation->OutDescNum) { Addr.bit32 =
     pStation->FmmuList[ pStation->OutDescList[0].Fmmu].s.LogicalAddr; }
     if (pStation->InDescNum)  { Addr.bit32 =
     pStation->FmmuList[pStation->InDescList[0].Fmmu].s.LogicalAddr; }

     //Get data length, if descriptors are present
     for (i=0; i<pStation->OutDescNum; i++)
     { DataSize[0] += pStation->OutDescList[i].Len; }
     for (i=0; i<pStation->InDescNum;  i++)
     { DataSize[1] += pStation->InDescList[i].Len; }

     //Set max. data length
     if ((DataSize[1]) && (DataSize[1] >= DataSize[0])) { Len = DataSize[1]; }
     if ((DataSize[0]) && (DataSize[0] >= DataSize[1])) { Len = DataSize[0]; }

     //Set command
     if (( pStation->OutDescNum) && ( pStation->InDescNum)){ Cmd = LRW_CMD; }
     if (( pStation->OutDescNum) && (!pStation->InDescNum)){ Cmd = LWR_CMD; }
     if ((!pStation->OutDescNum) && ( pStation->InDescNum)){ Cmd = LRD_CMD; }
     if ((!pStation->OutDescNum) && (!pStation->InDescNum)){ Cmd = NOP_CMD; }



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

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


void InitProcessTelegrams(void)
{
     //Loop through all enabled stations
     for (short i=0; i<__StationNum; i++)
                   if (!__pUserList[i].bDisable)
                        __InitProcessTelegram(&__pUserList[i]);
}
```

3.3 Header File ECAT*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,__WorkCnt)
```

This Inline-Macro is to get telegram information:

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

This Inline-Macro is to copy telegrams:

```
__EcatCpyTelegram(__pDstTel,__pSrcTel)

__pDstTel         Type: PECAT_TELEGRAM
__pSrcTel         Type: PECAT_TELEGRAM
```

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
```

3.4 Header File ECAT*64*SDODEF.H

This header file defines structures required for COE communication with the Library Expert Interface (for more detailed information see EtherCAT specification)

```
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;


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

3.5 Header File ECAT*64*EOEDEF.H

This header file defines structures required for EOE communication with the Library Expert Interface (for more detailed information see EtherCAT specification)

3.6 Header File ECAT*64*SIIDEF.H

This header file defines structures required for EEPROM (SII) Access, as well as parsing SII Category information, when using Library Expert interface (for more detailed information see EtherCAT specification).

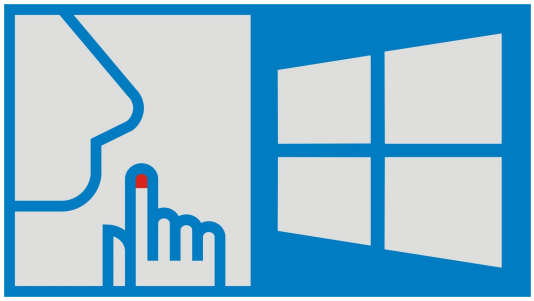
3.7 Header File ECAT*64*DCDEF.H

This header file defines structures required for Distributed Clock Access, when using Library Expert interface (for more detailed information see EtherCAT specification)

3.8 Header File ECAT64Control.H

This header file defines all macros for the ethernet core management. The inline function adjusts all required ethernet parameters for proper cyclic operation.

```
__inline ULONG __EcatSetEthernetMode(
                                    ULONG DevNum,     //Adapter Index
                                    ULONG Period,     //Sampling Period
                                    ULONG Cycles,     //Update Cycles
                                    BOOLEAN bDc)      //Distributed Clock Flag
```



3.9 Header File ECAT*64SILENT*DEF.H

The EtherCAT Master library supports the realtime silent mode. This means, that critical state changes (e.g. for DC drive management) can be done without Windows work load. This header file defines macros to use silent mode within a EtherCAT project.

This inline function has to be used when changing from PRE_OP state to OP state (as final state) at the windows part:

```
__inline ULONG __Ecat64SilentChangeState(
                                    PETH_STACK pStack,
                                    ULONG Period,
                                    USHORT AlStateFinal)
```

This inline function has to be placed inside the realtime task, following directly after __fpEcatEnter:

```
__inline void __SilentStateTransition(
                                    PSTATION_INFO pStationList,
                                    USHORT StationNum,
                                    PSTATE_OBJECT pStateObject)
```

(see sample SILENTSTATE)

4 EtherCAT Library Beginner 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.

4.1.1 Sha*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.

```
ULONG Sha64EcatGetVersion (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);
```

4.1.2 Sha*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().

```
ULONG Sha64ECatCreate (PECAT_PARAMS);
```

Sample:

```
//Required ECAT parameters
ECAT_PARAMS EcatParams;
EcatParams.FixedAddress = 1001;
EcatParams.LogicalAddress = 0x00010000;
EcatParams.EthParams.dev_num = 0;
EcatParams.EthParams.period = 1000;
EcatParams.SyncCycles = 2
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. 1000µsec) for RX, TX and ERR tasks. Cyclic Ethtercat telegrams will be handled by a synchronizing period:

*EcatParams.EthParams.period * EcatParams.SyncCycles*

(e.g. 1000µsec * 2 = 2msec)

4.1.3 Sha*64*EcatDestroy

This function closes the EtherCAT communication.

```
ULONG Sha64ECatDestroy(PECAT_PARAMS);
```

4.1.4 Sha*64*EcatEnable

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

```
ULONG Sha64ECatEnable(PECAT_PARAMS);
```

Note: Library Expert Routines implemented in ShaEcatEnable with following sequence:

```
//Clear CRC Fault Counter and reset devices
Ecat64ResetDevices();

//Change state to INIT
Ecat64ChangeAllStates(AL_STATE_INIT);

Ecat64InitStationAddresses();
Ecat64InitFmmus();
Ecat64InitSyncManagers();

//Change state to PRE OPERATIONAL
Ecat64ChangeAllStates(AL_STATE_PRE_OP);

//Init PDO assignment
Ecat64PdoAssignment();

//Init DC
Ecat64ReadDcLocalTime();
Ecat64CompDcOffset();
Ecat64CompDcPropDelay();
Ecat64CompDcDrift(&CompLoops);
Ecat64DcControl();

//Init process telegrams (required for AL_STATE_SAFE_OP);
Ecat64InitProcessTelegrams();

//Change state to SAFE OPERATIONAL / OPERATIONAL
Ecat64ChangeAllStates(AL_STATE_SAFE_OP); Sleep(500);
Ecat64ChangeAllStates (AL_STATE_OP);     Sleep(100);
```

4.1.5 Sha*64*EcatEnableDC

This function enables the EtherCAT station list with distributed clock management and must follow the function ShaEcatCreate.

```
ULONG Sha64ECatEnableDC(PECAT_PARAMS, PSTATE_OBJECT);
```

Note: Library Expert Routines implemented by ShaEcatEnableDC with following sequence:

```
//Clear CRC Fault Counter and reset devices
Ecat64ResetDevices();

//Change state to INIT
Ecat64ChangeAllStates(AL_STATE_INIT);

Ecat64InitStationAddresses();
Ecat64InitFmmus();
Ecat64InitSyncManagers();

//Change state to PRE OPERATIONAL
Ecat64ChangeAllStates(AL_STATE_PRE_OP);

//Init PDO assignment
Ecat64PdoAssignment();

//Init DC
Ecat64ReadDcLocalTime();
Ecat64CompDcOffset();
Ecat64CompDcPropDelay();
Ecat64CompDcDrift(&CompLoops);
Ecat64DcControl();

//Init process telegrams (required for AL_STATE_SAFE_OP);
Ecat64InitProcessTelegrams();

//Change state to SAFE OPERATIONAL / OPERATIONAL
Ecat64CyclicStateControl(pStateObject, AL_STATE_SAFE_OP); Sleep(500);
Ecat64CyclicStateControl(pStateObject, AL_STATE_OP);      Sleep(100);
```

4.1.6 Sha*64*EcatDisable

This function disables the EtherCAT station list

```
ULONG Sha64ECatDisable(PECAT_PARAMS);
```

Note: Library Expert Routines implemented in Sha64EcatDisable with following sequence:

```
//Change state to INIT
Ecat64ChangeAllStates(AL_STATE_INIT); Sleep(100);
```

4.1.7 Sha*64*EcatDisableDC

This function disables the EtherCAT station list with distributed clock management

```
ULONG Sha64ECatDisableDC(PECAT_PARAMS, PSTATE_OBJECT);
```

Note: Library Expert Routines implemented in Sha64EcatDisableDC with following sequence:

```
//Change state to INIT
Ecat64CyclicStateControl(pStateObject, AL_STATE_INIT); Sleep(100);
```

Sample:

```
#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)
{
      //Call enter wrapper function
      __EcatState = __fpEcatEnter(
                              __pSystemStack,
                              __pSystemList,
                              (USHORT)__StationNum,
                              &__StateObject);

      //Check operation state and decrease ready count
      if (__EcatState == ECAT_STATE_ACCESS) { __ReadyCnt = SYNC_CYCLES; }
      if (__EcatState == ECAT_STATE_READY)  { __ReadyCnt--; }

      //Check ready count
      if (__ReadyCnt == 1)    {
            //*********************************
            //Do the logical station operation
            //*********************************

            __UpdateCnt++;
      }
      //Call exit function
      __fpEcatExit();

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

```
void main(void)
{
	//Required ECAT parameters
	ECAT_PARAMS EcatParams;
	memset(&EcatParams, 0, sizeof(ECAT_PARAMS));
	EcatParams.PhysAddr = DEFAULT_PHYSICAL_ADDRESS;
	EcatParams.LogicalAddr = DEFAULT_LOGICAL_ADDRESS;
	EcatParams.SyncCycles = SYNC_CYCLES;
	EcatParams.EthParams.dev_num = 0;
	EcatParams.EthParams.period = REALTIME_PERIOD;
	EcatParams.EthParams.fpAppTask = AppTask;

	//Create ECAT realtime core
	if (ERROR_SUCCESS == Sha64EcatCreate(&EcatParams))
	{
		//Init global 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
#ifdef DC_CONTROL
		if (ERROR_SUCCESS == Sha64EcatEnableDC(&EcatParams, &__StateObject))
#else
		if (ERROR_SUCCESS == Sha64EcatEnable(&EcatParams))
#endif
		{
			//Do a check loop
			printf("\nPress any key ...\n");
			while (!kbhit())
			{
				//Display realtime information
				printf("Loop Count: %i, Update Count: %i\r",
					__LoopCnt, __UpdateCnt);

				//Do some delay
				Sleep(100);
			}

			//Disable Stations
#ifdef DC_CONTROL
			Sha64EcatDisableDC(&EcatParams, &__StateObject);
#else
			Sha64EcatDisable(&EcatParams);
#endif
		}
		//Destroy ECAT core
		Sha64EcatDestroy(&EcatParams);
	}
}
```

5 EtherCAT Library (Expert Code)

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

5.1 EtherCAT Command Functions

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

5.1.1 Send EtherCAT Command

This is an universal function for sending EtherCAT commands

```
ULONG Result = Ecat64SendCommand(
                                UCHAR       Cmd,
                                USHORT      Adp,
                                USHORT      Ado,
                                USHORT      DataSize,
                                PUCHAR      pData)
```

Sample:

```
//Send ethercat command
ULONG Result = Ecat64SendCommand(APWR_CMD, 0xFFFE, 0x120, 2,
      (PUCHAR)"\x01\x00");
```

5.1.2 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 = Ecat64ResetDevices(void);
```

5.1.3 Clear Error Counters

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

```
ULONG Result = Ecat64CheckErrorCounters(BOOLEAN bReset);
```

5.1.4 Read DL Information

Read DL information into station list

```
ULONG Result = ECat64ReadDlInfo(void);
```

5.1.5 Read DL Status

Read DL Status information into station list

```
ULONG Result = Ecat64ReadDlStatus(void);
```

5.1.6 Read/Write DL Control

Read/Write DL Control information into station list

```
ULONG Result = Ecat64CheckDlControl(BOOLEAN bWrite);
```

5.1.7 Read PDI Control

Read PDI Control information into station list

```
ULONG Result = Ecat64ReadPDIControl(void);
```

5.1.8 Read PDI Configuration

Read PDI Configuration information into station list

```
ULONG Result = Ecat64ReadPDIConfig(void);
```

5.1.9 Init Station Addresses

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

```
ULONG Result = Ecat64InitStationAddresses(USHORT PhysStartAddress);
```

5.1.10 Init Alias Addresses

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

```
ULONG Result = Ecat64InitAliasAddresses(void);
```

5.1.11 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 = Ecat64InitSyncManagers(void);
```

5.1.12 Configure FMMU Management (parallel)

The parallel FMMU Manager maps the cyclic data into a logical datagram of each station.

```
ULONG Result = Ecat64InitFmmus (void);
```

Note: Logical Addessing Sheme

The EtherCAT Realtime Library provides an integrated logical addressing sheme. Thereby all 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);
}
```

While each station is given it’s own logical datagram, each datagram command differs due to it’s data direction, described by the FMMUs.

5.1.13 Init process Telegrams (parallel)

The parallel cyclic data exchange is based on one logical datagram for each station. The function Ecat64InitProcessTelegram setup a logical datagram due to the FMMU mapping and the DATA_DESC descriptors as following:

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

//Get logical address of first descriptor, if descriptors are present
if (pStation->OutDescNum) { Addr.bit32 =
            pStation->FmmuList[pStation->OutDescList[0].Fmmu].s.LogicalAddr; }
if (pStation->InDescNum)  { Addr.bit32 =
            pStation->FmmuList[pStation->InDescList[0].Fmmu].s.LogicalAddr; }

//Get data length, if descriptors are present
for (i=0; i<pStation->OutDescNum; i++)
      { DataSize[0] += pStation->OutDescList[i].Len; }
for (i=0; i<pStation->InDescNum;  i++)
      { DataSize[1] += pStation->InDescList[i].Len; }

//Set max. data length
if ((DataSize[1]) && (DataSize[1] >= DataSize[0])) { Len = DataSize[1]; }
if ((DataSize[0]) && (DataSize[0] >= DataSize[1])) { Len = DataSize[0]; }

//Set command
if (( pStation->OutDescNum) && ( pStation->InDescNum)) { Cmd = LRW_CMD; }
if (( pStation->OutDescNum) && (!pStation->InDescNum)) { Cmd = LWR_CMD; }
if ((!pStation->OutDescNum) && ( pStation->InDescNum)) { Cmd = LRD_CMD; }
if ((!pStation->OutDescNum) && (!pStation->InDescNum)) { Cmd = NOP_CMD; }

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

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


void Ecat64InitProcessTelegrams(void)
{
      //Loop through all enabled stations
      for (short i=0; i<__StationNum; i++)
      if (!__pUserList[i].bDisable)
            __InitProcessTelegram(&__pUserList[i]);
}
```

5.1.14 Configure FMMU Management (combined)

The combined FMMU Manager maps the cyclic data into one logical datagram.

```
ULONG Result = Ecat64InitFmmusCombined (void);
```

Note: Logical Addessing Sheme

```
__inline void __GetLogicalDataOffs(
                              PSTATION_INFO pStation,
                              ULONG LogicalStartAddr)
{
      //Get FMMU index from data descriptor list
      UCHAR InputIndex  = pStation->InDescList[0].Fmmu;
      UCHAR OutputIndex = pStation->OutDescList[0].Fmmu;

      //Get logical address of first descriptor, if descriptors are present
      if (pStation->InDescNum)
      { pStation->ExtInfo.LogicalDataOffs[FMMU_INPUT] =
        pStation->FmmuList[InputIndex].s.LogicalAddr  - LogicalStartAddr; }
      if (pStation->OutDescNum)
      { pStation->ExtInfo.LogicalDataOffs[FMMU_OUTPUT] =
        pStation->FmmuList[OutputIndex].s.LogicalAddr - LogicalStartAddr; }
}

//Loop through all enabled stations
for (ULONG i=0; i <StationNum; i++)
{
      //Get station pointer
      PSTATION_INFO pStation = (PSTATION_INFO)STATION_PTR(i);

      //Calculate all logical addresses for station FMMUs
      LogicalAddr[FMMU_INPUT]  +=
      __EcatCalcStationAddresses(pStation, LogicalAddr[FMMU_INPUT], TRUE);
      LogicalAddr[FMMU_OUTPUT] +=
      __EcatCalcStationAddresses(pStation, LogicalAddr[FMMU_OUTPUT], FALSE);

      //Get logical data offset (input/output) for this station
      __GetLogicalDataOffs(pStation, LogicalStartAddr);
}
```

5.1.15 Init process Telegrams (combined)

The combined cyclic data exchange is based on one logical datagrams for all stations. The function Ecat64InitProcessTelegramCombined setup a logical datagram due to the FMMU mapping and the DATA_DESC descriptors as following:

```
ULONG Ecat64InitProcessTelegramsCombined(ULONG LogicalAddr)
```

Note: Datagram setup

```
//Loop through all enabled stations
//Get data length, if descriptors are present
USHORT Offs[2] = { 0 };
for (short i=0; i<__StationNum; i++)
{
      PSTATION_INFO pStation = (PSTATION_INFO)&__pUserList[i];
      if (!pStation->bDisable)
      {
            Offs[0] +=__GetStationDataLength(pStation, TRUE);
            Offs[1] +=__GetStationDataLength(pStation, FALSE);
      }
}

//Set command address
//Set command data length
TYPE32 Addr = { LogicalAddr };
USHORT Len  = (Offs[0] > Offs[1]) ? Offs[0] : Offs[1];

//Set cyclic telegram for first station
PSTATION_INFO pStation = (PSTATION_INFO)&__pUserList[0];
__EcatSetCyclicTelegram(&pStation->TxTel, (UCHAR)pStation->Index,
                  LRW_CMD, Addr.bit16[0], Addr.bit16[1], Len, NULL, 0x0000);

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

5.1.16 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 = Ecat64PdoAssignment (void);
```

This command proceeds following actions:

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

5.1.17 Watchdog Enable

Enables/Disables all watchdog controls of the station list

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

5.2 EtherCAT 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.

Note: While in state INIT or PRE_OP, the realtime task is not running.

5.2.1 Read AL Status

Read AL status of all stations: APRD Offs 0x130

```
ULONG Result = ECat64ReadAlStatus(void);
```

5.2.2 Change All States (acyclic)

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 = ECat64ChangeAllStates(UCHAR State);
```

5.2.3 Change State By Node Address (acyclic)

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 = ECat64ChangeStatesByNodeAddress(
                                        UCHAR  State,
                                        USHORT StationAddress);
```

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())
            {
                  //Init process telegrams
                  InitProcessTelegrams();

                  if (ERROR_SUCCESS == EcatChangeAllStates(AL_STATE_SAFE_OP))
                  {
                        //Change state to OPERATIONAL
                        if (ERROR_SUCCESS == EcatChangeAllStates(AL_STATE_OP))
                        {
                              …
```

Note:

After state PRE_OP, the acyclic state change should only be used when working without Distributed Clocks (DC). Otherwise the pulse wide modulation for the reference clock cannot be controlled.

5.2.4 Cyclic State Change

For Distributed Clock (DC) control it's required to use the cyclic state change for changing the state to AL_STATE_SAFE_OP and AL_STATE_ OP.

```
ULONG Result = Ecat64CyclicStateControl(
                                    PSTATE_OBJECT pStateObject
                                    UCHAR State)
```

Sample:

```
STATE_OBJECT __StateObject = { 0 }; //Cyclic state object


void static AppTask (PVOID)
{
   //Check if initialization has been done
   if (__bInitDone == FALSE)
       return;

   //Call enter wrapper function
   __EcatState = __fpEcatEnter(
             __pSystemStack,
             __pSystemList,
             (USHORT)__StationNum,
             &__StateObject);
…
}


void main (void)
{
   //Change state to INIT (acyclic)
   if (ERROR_SUCCESS == Ecat64ChangeAllStates(AL_STATE_INIT))
   {
       //Reset devices
       Ecat64ResetDevices();

       //Set fixed station addresses and
       //Init FMMUs and SYNCMANs
       if (ERROR_SUCCESS == Ecat64InitStationAddresses(EcatParams.PhysAddr))
          if (ERROR_SUCCESS == Ecat64InitFmmus(EcatParams.LogicalAddr))
          if (ERROR_SUCCESS == Ecat64InitSyncManagers())
          {
            //Change state to PRE OPERATIONAL
            //Init PDO assignment
            if (ERROR_SUCCESS == Ecat64ChangeAllStates(AL_STATE_PRE_OP))
            if (ERROR_SUCCESS == Ecat64PdoAssignment())
            {
```

```
//Drift compensation delay
ULONG CompLoops = 1000;

    //Init DC immediately after cyclic operation has started
    //and get static master drift per msec (nsec unit)
    if (ERROR_SUCCESS == Ecat64ReadDcLocalTime())
    if (ERROR_SUCCESS == Ecat64CompDcOffset())
    if (ERROR_SUCCESS == Ecat64CompDcPropDelay())
    if (ERROR_SUCCESS == Ecat64CompDcDrift(&CompLoops))
    if (ERROR_SUCCESS == Ecat64DcControl())
    {
      //Init process telegrams
      InitProcessTelegrams ();

      //**********************
      //Start cyclic operation
      //**********************

      //Change state to SAFE OPERATIONAL cyclic
      if (ERROR_SUCCESS == Ecat64CyclicStateControl(
                                          &__StateObject,
                                          AL_STATE_SAFE_OP))
      {
        //Do some delay
        Sleep(500);

        //Change state to SAFE OPERATIONAL cyclic
        if (ERROR_SUCCESS == Ecat64CyclicStateControl(
                                                  &__StateObject,
                                                  AL_STATE_OP))
        {
               //Do some delay
               Sleep(100);
      …

               //**********************
               //Stop cyclic operation
               //**********************
               Ecat64CyclicStateControl(&__StateObject, AL_STATE_INIT);
```

5.2.5 Silent State Change

The state change between PRE_OP to OP is strongly time critical, since the timing measurement for distributed clock (DC) takes place. Since the realtime jitter behaviour depends on the Windows work load, it's important to reduce this factor. Thereby on silent mode, Windows is blocked for a period of time. The following functions are defined as inline macros in the header file ECAT64SILENTDEF.H.

Request silent state change (within Windows code)

```
__inline ULONG __Ecat64SilentChangeState(
                                    PETH_STACK pStack,
                                    ULONG Period,
                                    USHORT AlStateFinal)
```

Do silent state transition (within Realtime code)

```
__inline void __SilentStateTransition(
                               PSTATION_INFO pStationList,
                               USHORT StationNum,
                               PSTATE_OBJECT pStateObject)
```

Sample

```
STATE_OBJECT __StateObject = { 0 }; //Cyclic state object


//*********************
SILENT_INIT_ELEMENTS();
//*********************


void static AppTask(PVOID)
{
      //Call enter wrapper function
      __EcatState = __fpEcatEnter(
                              __pSystemStack,
                              __pSystemList,
                              (USHORT)__StationNum,
                              &__StateObject);

      //Control state silent
      __SilentStateTransition(
                        __pSystemList,
                        __StationNum,
                        &__StateObject);

      //Check operation state and increase ready count
      if (__EcatState == ECAT_STATE_ACCESS) { __ReadyCnt = SYNC_CYCLES; }
      if (__EcatState == ECAT_STATE_READY)  { __ReadyCnt--; }

      //Check operation state and increase ready count
      if (__ReadyCnt == 1)
      {
            //*******************
            //Do logic operation
            //*******************

            //Update counter
            __UpdateCnt++;
      }

      //Call exit function
      __fpEcatExit();

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

```
void main(void)
{
      …

    //Change state to INIT
    if (ERROR_SUCCESS == Ecat64ChangeAllStates(AL_STATE_INIT))
    {
         //Reset devices
         Ecat64ResetDevices();
         Ecat64SendCommand(BWR_CMD, 0x0000,  0x300, 8, Data);

         //Set fixed station addresses and
         //Init FMMUs and SYNCMANs
         if (ERROR_SUCCESS == Ecat64InitStationAddresses(EcatParams.PhysAddr))
             if (ERROR_SUCCESS == Ecat64InitFmmus(EcatParams.LogicalAddr))
             if (ERROR_SUCCESS == Ecat64InitSyncManagers())
             {
                  //Change state to PRE OPERATIONAL
                  //Init PDO assignment
                  if (ERROR_SUCCESS == Ecat64ChangeAllStates(AL_STATE_PRE_OP))
                  if (ERROR_SUCCESS == Ecat64PdoAssignment())
                  {
                      ULONG Loops = 1000;

                      //Init DC immediately after cyclic operation has started
                      //and get static master drift per msec (nsec unit)
                      if (ERROR_SUCCESS == Ecat64ReadDcLocalTime())
                      if (ERROR_SUCCESS == Ecat64CompDcOffset())
                      if (ERROR_SUCCESS == Ecat64CompDcPropDelay())
                      if (ERROR_SUCCESS == Ecat64CompDcDrift(&Loops))
                      if (ERROR_SUCCESS == Ecat64DcControl())
                      {
                          //Init process telegrams (required for AL_STATE_SAFE_OP)
                          Ecat64InitProcessTelegrams();

                          //*********************************
                          //Change state to OPERATIONAL silent
                          //*********************************
                          __Ecat64SilentChangeState(
                                           __pUserStack,
                                           REALTIME_PERIOD,
                                           AL_STATE_OP);

                          //Do a check loop
                          printf("\nPress any key ...\n");
                          while (!kbhit())
                          {
                                //Display TX and RX information
                                printf("Update Count: %i, Status: 0x%04x\r",
                                __UpdateCnt,
                                __TxPdoMap.StatusWord);

                                //Do some delay
                                Sleep(100);
                          }
```

```
                        //**************************
                        //Change state to INIT silent
                        //**************************
                        __Ecat64SilentChangeState(
                                        __pUserStack,
                                        REALTIME_PERIOD,
                                        AL_STATE_INIT);
                }
            }
            //Destroy ECAT core
            Sha64EcatDestroy(&EcatParams);
        }
}
```

5.3 EtherCAT 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.

5.3.1 Initiate SDO Download Expedited Request

This function initiates a SDO Download Expedited Request

```
ULONG SHAAPI Ecat64SdoInitDownloadReq(
                              PSTATION_INFO pStation,
                              USHORT SdoIndex,
                              UCHAR SdoSubIndex,
                              ULONG SdoSize,
                              PUCHAR pSdoData)
```

5.3.2 Initiate SDO Download Expedited Response

This function initiates a SDO Download Expedited Response

```
ULONG Ecat64SdoInitDownloadResp(PSTATION_INFO pStation);
```

5.3.3 Initiate SDO Upload Expedited Request

This function initiates a SDO Upload Expedited Request

```
ULONG Ecat64SdoInitUploadReq(
                        PSTATION_INFO pStation,
                        USHORT SdoIndex,
                        UCHAR  SdoSubIndex);
```

5.3.4 Initiate SDO Download Expedited Response

This function initiates a SDO Download Expedited Response

```
ULONG Ecat64SdoInitUploadResp(
                        PSTATION_INFO pStation,
                        PULONG pSdoSize,
                        PUCHAR pSdoData)
```

Sample:

```
//Get station pointer
PSTATION_INFO pStation = (PSTATION_INFO)&__pUserList[3];

//SDO Download Request/Response
ULONG Error = 0;
ULONG SdoData = 0;
Error = Ecat64SdoInitDownloadReq( pStation, 0x1c13, 0x00, 1, (PUCHAR)&SdoData);
Error = Ecat64SdoInitDownloadResp(pStation);

//SDO Upload Request/Response
ULONG SdoSize = 0;
UCHAR SdoBuffer[MAX_ECAT_DATA] = { 0 };
Error = Ecat64SdoInitUploadReq( pStation, 0x1a00, 0x01);
Error = Ecat64SdoInitUploadResp(pStation, &SdoSize, SdoBuffer);
```

5.4 EtherCAT 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.

5.4.1 Write command to mailbox (sequential)

```
ULONG Result = Ecat64MailboxWrite(
                        PSTATION_INFO pStation,
                        PUCHAR pData,
                        USHORT DataSize,
                        UCHAR MailboxType)
```

5.4.2 Read command from mailbox (sequential)

```
ULONG Result = Ecat64MailboxRead(
                        PSTATION_INFO pStation,
                        PUCHAR pData)
```

5.4.3 Check mailbox for pending response (sequential)

```
ULONG Result = Ecat64MailboxCheck(PSTATION_INFO pStation)
```

5.4.4 Write command to mailbox (parallel)

```
ULONG Result = Ecat64MailboxWriteAll(
                              PMAILBOX_INFO pInfoList,
                              ULONG InfoNum,
                              UCHAR MailboxType)
```

5.4.5 Read command from mailbox (parallel)

```
ULONG Result = Ecat64MailboxReadAll(
                              PSTATION_INFO pStationList,
                              PMAILBOX_INFO pInfoList,
                              ULONG InfoNum)
```

5.4.6 Check mailbox for pending response

This function checks a mailbox for pending response

```
ULONG Result = Ecat64MailboxCheckAll(
                              PMAILBOX_INFO pInfoList,
                              ULONG InfoNum)
```

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);
```

5.5 EtherCAT 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.

5.5.1 Read SII Data

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

```
ULONG Result = Ecat64SiiRead(
                        PSTATION_INFO pStation,
                        PUCHAR pData,
                        USHORT DataSize,
                        USHORT Offs)
```

5.5.2 Write SII Data

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

```
ULONG Result = Ecat64SiiWrite(
                        PSTATION_INFO pStation,
                        PUCHAR pData,
                        USHORT DataSize,
                        USHORT Offs)
```

5.5.3 Reload SII Data

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

```
ULONG Result = Ecat64SiiReload(
                        PSTATION_INFO pStation,
                        USHORT DataSize,
                        USHORT Offs)
```

5.5.4 Get Category String

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

```
ULONG SHAAPI Ecat64GetCategoryGeneral(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pGeneral)
```

5.5.5 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 = Ecat64GetCategoryString(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                char* pszStr,
                                ULONG StrIndex)
```

5.5.6 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 = Ecat64GetCategorySyncman(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pSyncman,
                                ULONG SyncmanIndex)
```

5.5.7 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 = Ecat64GetCategoryFmmu(
                                PUCHAR pCatArea,
                                ULONG CatAreaSize,
                                PUCHAR pFmmu,
                                ULONG FmmuIndex)
```

5.5.8 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 = Ecat64GetCategoryPdo(
                                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);
```

5.6 EtherCAT 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.
The distributed clock functions are to be used directly after change ti PRE-OPERATIONAL state.

5.6.1 DC Local Time

This function latches out the local time of all stations.

```
ULONG Result = Ecat64ReadDcLocalTime(VOID);
```

5.6.2 DC Propagation Delay Compensation

This function compensates the propagation delay for the stations relations

```
ULONG Result = Ecat64CompDcPropDelay(VOID);
```

5.6.3 DC Offset Compensation

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

```
ULONG Result = Ecat64CompDcOffset(VOID);
```

5.6.4 DC Drift Compensation

This function compensates the drift of the DC station clock by writing ARMW commands at least 1000 Loops.

```
ULONG Result = Ecat64CompDcDrift(PULONG pCompLoops);
```

5.6.5 DC Quality Check

This function checks the quality of DC synchronisation. It returns the max. system time difference in [nsec] among all stations. Additionally the individual system time difference is written to DC_LOCAL_TIME structure of each station.

```
ULONG Result = Ecat64CheckDcQuality(PULONG pMaxSysTimeDiff);
```

5.6.6 Read DC Synchronisation Information

This function reads the complete DC_SYNC_INFO structure for further DC processing

```
ULONG Result = Ecat64ReadDcSyncInfo(VOID);
```

5.6.7 DC Sync Control (indirect)

This function enables the synchronisation output signal, due to the DC configuration in ECATDEVICE[name].PAR.

```
ULONG Result = Ecat64DcControl(VOID);
```

5.6.8 DC Sync Control (direct)

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

```
ULONG SHAAPI Ecat64SyncControl(
                        PSTATION_INFO pStation,
                        ULONG Sync0CycleTime,
                        ULONG Sync1CycleTime,
                        ULONG Sync0CycleShift,
                        ULONG Sync1CycleShift,
                        BOOLEAN bSync0Pulse,
                        BOOLEAN bSync1Pulse,
                        BOOLEAN bSyncPdiCtrl)
```

Sample:

```
      Ecat64SyncControl(
                  &__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.

6 Native Device Configuration

Usualy device information is provided by a corresponding ESI configuration file. Since the development of software with the EtherCAT Master Library has special needs for programming, the ESI file must be parsed and translated into a native format. Therefore the EtherCAT Master Library provides a configuration file called **ECATDEVICE[name].PAR**, whitch has to be located in the current execution directory after installation. The ECATDEVICE[name].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
[OPMODE]
30 07 40 0d 03 00 40 42 0f 00 f0 d8 ff ff 00 00 00 00 ff ff ff ff
```

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.

6.1 Section [NAME]

This section contains the name of the device:

```
[NAME]
EL3102
```

6.2 Section [VENDOR]

This section contains the vendor ID of the device:

```
[VENDOR]
00000002
```

6.3 Section [CODE]

This section contains the product code of the device:

```
[CODE]
0C1E3052
```

6.4 Section [REVISION]

This section contains the revision number of the device:

```
[REVISION]
00100000
```

Note: If the revision value is not present, all revisions are considered

6.5 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 0x02: mailbox
Direction	0x0004	Unsigned2	RW	R	0x00: area shall be read from the master 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 0x01: DLS-user event is active (when area was accessed and is no longer locked)
Watchdog enable	0x0004	Unsigned1	RW	R	0x00: watchdog disabled 0x01: watchdog enabled
reserved	0x0004	Unsigned1	RW	R	0x00
Write event	0x0005	Unsigned1	R	R	0x00: no write event 0x01: write event
Read event	0x0005	Unsigned1	R	R	0x00: no read event 0x01: read event

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

6.6 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 0x01: entity will be used for read service
Write enable	0x000B	Unsigned1	RW	R	0x00: entity will be ignored for write service 0x01: entity will be used for write service
reserved	0x000B	Unsigned6	RW	R	0x00
Enable	0x000C	Unsigned1	RW	R	0x00: entity not active 0x01: entity active
reserved	0x000C	Unsigned15	RW	R	0x0000
reserved	0x000E	WORD	RW	R	0x0000

6.7 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
```

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 0x01: size of Data in Data Set Size specified
	Transfer Type	Unsigned1	0x01: Expedited transfer
	Data Set Size	Unsigned2	0x00: 4 Octet Data 0x01: 3 Octet Data 0x02: 2 Octet Data 0x03: 1 Octet Data
	Complete Access	Unsigned1	0x00
	Command	Unsigned3	0x01: Initiate Download Request

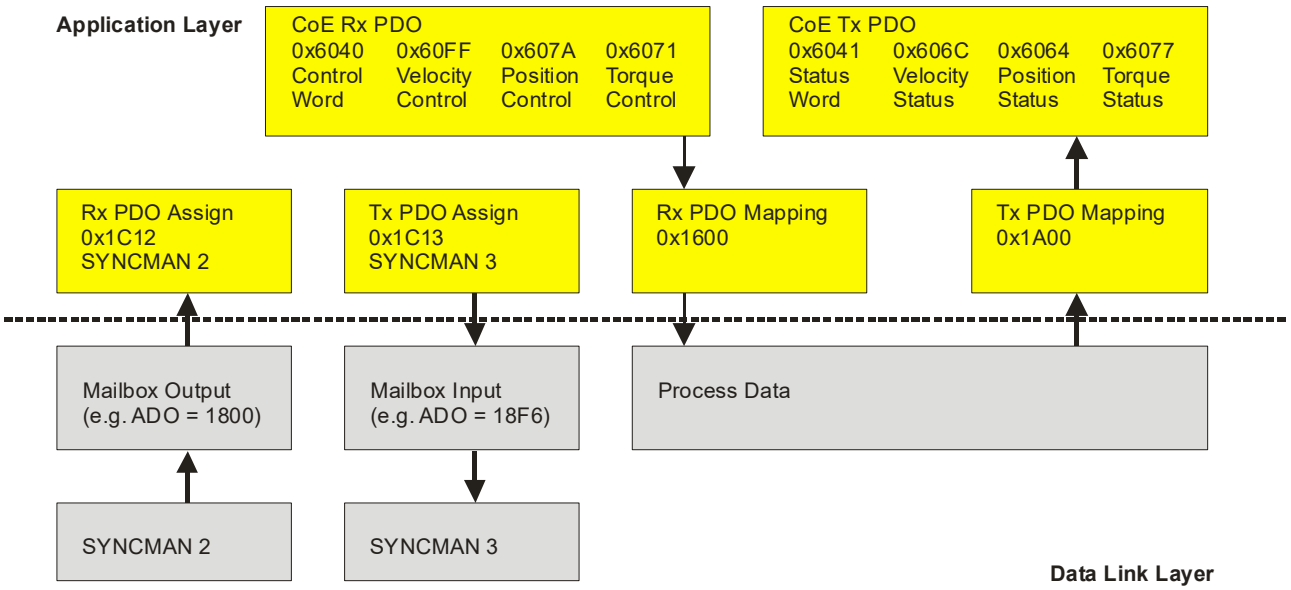
Sample:

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)

6.7.1 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



RX-PDO Mapping				TX-PDO Mapping			
Index	**Variables**	**Mapping**	**Datatyp**	**Index**	**Variables**	**Mapping**	**Datatyp**
0x1600	Controlword	0x6040	WORD	0x1A00	Statusword	0x6041	WORD
	Mode of Operation (Control)	0x6060	BYTE		Mode of Operation (Status)	0x6061	BYTE
	Halt Option Code	0x605D	WORD		Error Code	0x603F	WORD
	Target position	0x607A	DWORD		Position Actual Value (int)	0x6063	DWORD
					Position Actual Value	0x6064	DWORD
					Position Windows	0x6067	DWORD
	Target Velocity	0x6042	WORD		Velocity Actual Value	0x6044	WORD
	Target Velocity	0x60FF	DWORD		Velocity Actual Value	0x606C	DWORD
	Velocity Offset	0x6013	DWORD		Velocity Offset	0x60B1	DWORD
					Velocity Demand Value	0x606B	DWORD
	Target Torque	0x6071	DWORD		Torque Actual value	0x6077	DWORD
	Torque Offset	0x60B2	DWORD				
	Touch Probe (Control)	0x60B8	WORD		Touch Probe (Status)	0x60B9	WORD
					Touch Probe 1 Positive Edge	0x60BA	DWORD
					Touch Probe 1 Negative Edge	0x60BB	DWORD

6.8 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
```

6.9 Section [OPMODE]

This section contains the Distributed Clock configuration of the device: This configuration is managed by the DC Configurator of the ECATVERIFY Software.

```
[OPMODE]
30 07 40 0d 03 00 40 42 0f 00 f0 d8 ff ff 00 00 00 00 ff ff ff ff
```

Meaning (e.g.):

```
Control:          30 07       = 0x0730

CycleTimeSync0:   40 0d 03 00 =   200.000 [nsec]

CycleTimeSync1:   40 42 0f 00 = 1.000.000 [nsec]

ShiftTimeSync0:   f0 d8 ff ff =  - 10.000 [nsec]

ShiftTimeSync1:   00 00 00 00 =         0 [nsec]

CycleFactorSync0: ff ff       = - 1

CycleFactorSync1: ff ff       = - 1
```

7 Realtime Operation

After changing the state to SAFE OPERATIONAL, the cyclic operation is active and the realtime task start to run. 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);
```

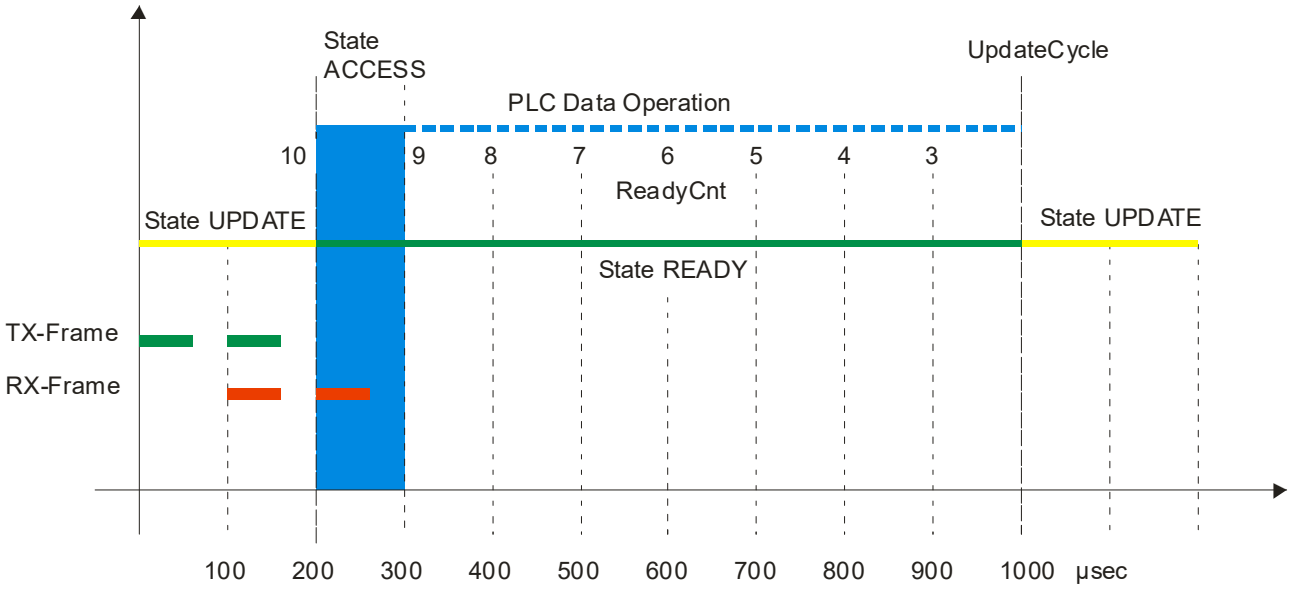
These wrapper functions are used to manage the cyclic EtherCAT data exchange at realtime. Also the station management, like ethernet frame update, error handling, DC synchronisation and stack management are handled by the wrapper functions. Since the Application task is running with a sampling period (e.g. 1000µsec), 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,           //Full bandwith for logical operation
      ECAT_STATE_ERROR,           //An update error occured
      ECAT_STATE_ACCESS           //Frame has been sent an is received
};
```

The Wrapper Functions require as parameter the Ethernet Stack pointer (e.g.__pSystemStack), the station list pointer (e.g.__pSystemList), the number of stations (e.g. __StationNum) and the reference of the cyclic state object (e.g. __StateObject). 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 within Windows Code)
PETH_STACK     __pSystemStack = NULL;     //Ethernet Core Stack
                                          //(used inside Realtime Task)
PSTATION_INFO  __pUserList = NULL;        //Station List
                                          //(used within Windows Code)
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 = 0;           //Initial Wrapper State
ULONG          __UpdateCnt = 0;           //Station Update Counter
ULONG          __LoopCnt = 0;             //Realtime Loop Counter
ULONG          __ReadyCnt = 0;            //Ready state counter
STATE_OBJECT   __StateObject = { 0 }      //Cyclic state object


//Create ECAT realtime core
if (ERROR_SUCCESS == Sha64EcatCreate(&EcatParams))
{
      //Init global elements
      __pUserStack      = EcatParams.EthParams.pUserStack;
      __pSystemStack    = EcatParams.EthParams.pSystemStack;
      __pUserList       = EcatParams.pUserList;
      __pSystemList     = EcatParams.pSystemList;
      __StationNum      = EcatParams.StationNum;
      __fpEcatEnter     = EcatParams.fpEcatEnter;
      __fpEcatExit      = EcatParams.fpEcatExit;
```

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:



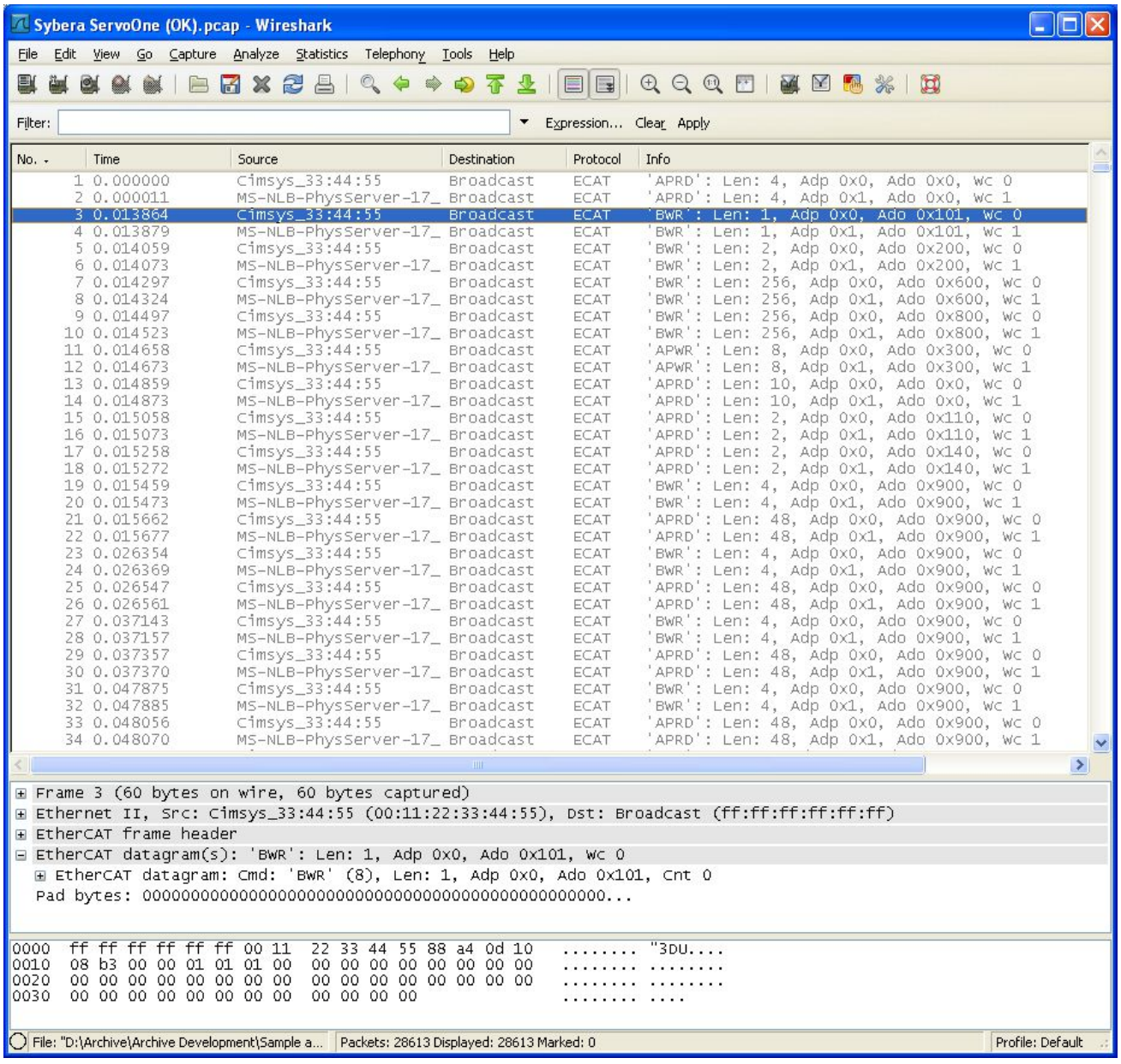
Sampling Period = 100 µsec
SyncCycles = 10

Note

For optimal operation, less SyncCycles are of advantage, since the bandwith is optimized. For single frame operation (all datagrams fit into a frame, only 1 SyncCycle is required. Then, the logical operation starts at state ECAT_STATE_ACCESS.

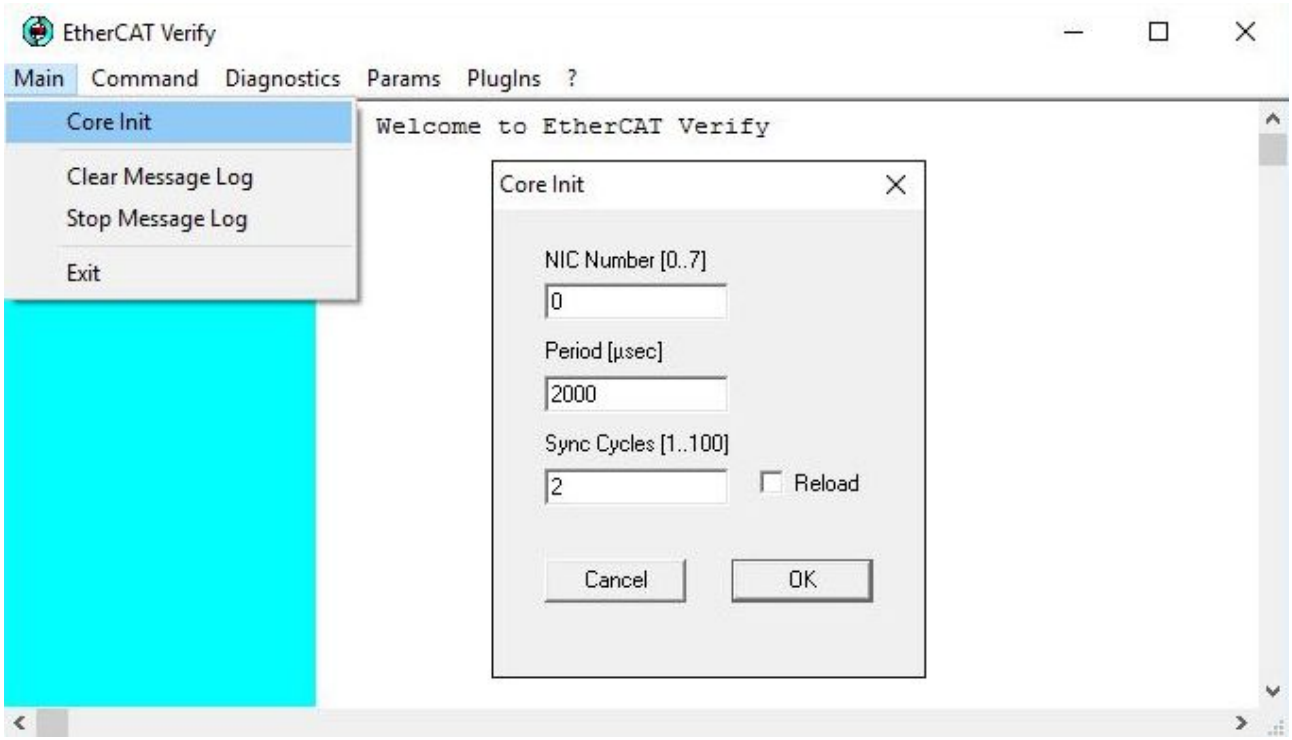
```
void static AppTask(PVOID)
{
      //Call enter wrapper function
      __EcatState = __fpEcatEnter(
                              __pSystemStack,
                              __pSystemList,
                              (USHORT)__StationNum,
                              &__StateObject);

      //Check operation state and decrease ready count
      if (__EcatState == ECAT_STATE_ACCESS) { __ReadyCnt = SYNC_CYCLES; }
      if (__EcatState == ECAT_STATE_READY)  { __ReadyCnt--; }

      //Check ready count
      if (__ReadyCnt == 1)
      {
            //*********************************
            //Do the logical station operation
            //*********************************

            __UpdateCnt++;
      }

      //Call exit function
      __fpEcatExit();

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

Sample Startup Wireshark Protocol:



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, and the synchronisation cycles of the realtime application task.



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:

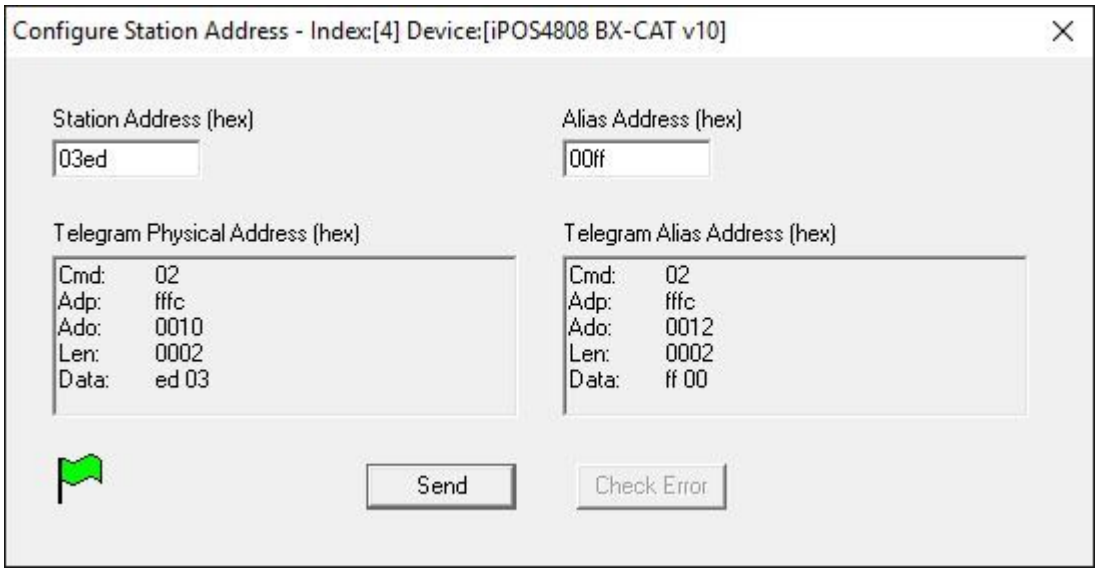
8.2 State Control Dialog

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.

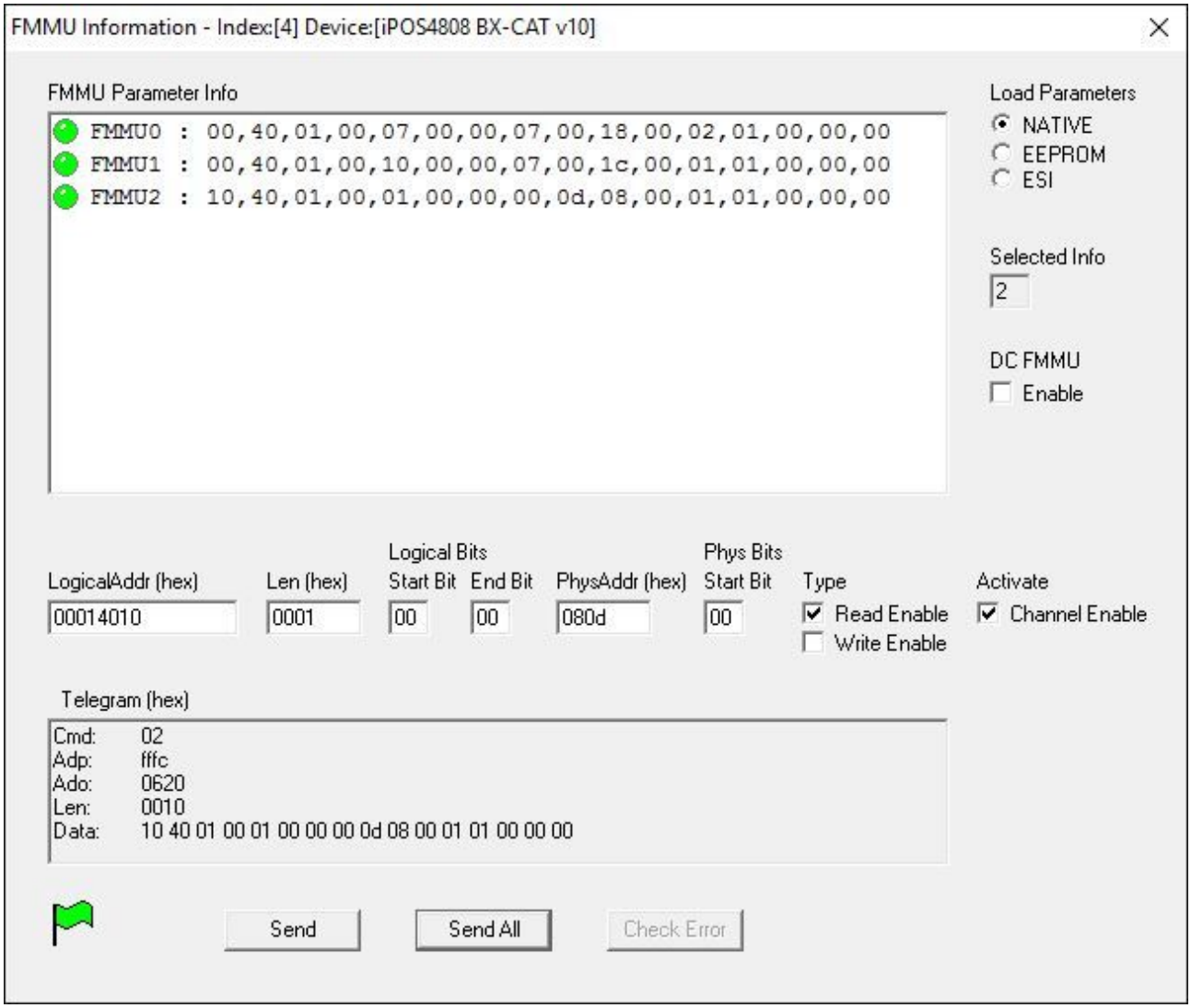
8.2.1 Configure Station Address

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



8.2.2 Configure FMMU Management

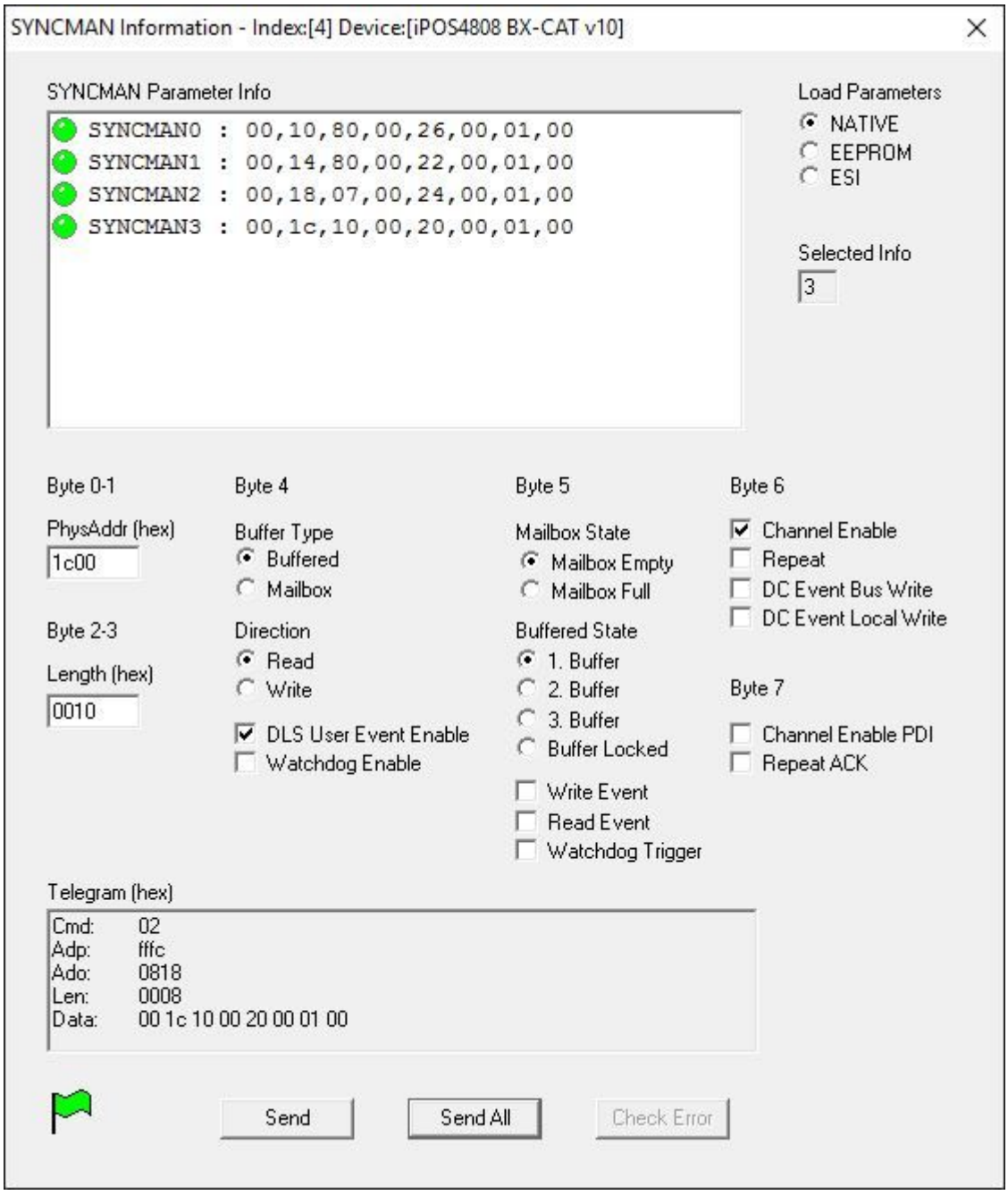
The FMMU ability dialog allows parsing XML information, EEPROM (SII) information and the Native format for configuration and provides information to all items (also described in the EtherCAT specification).



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.

8.2.3 Configure SYNC Management

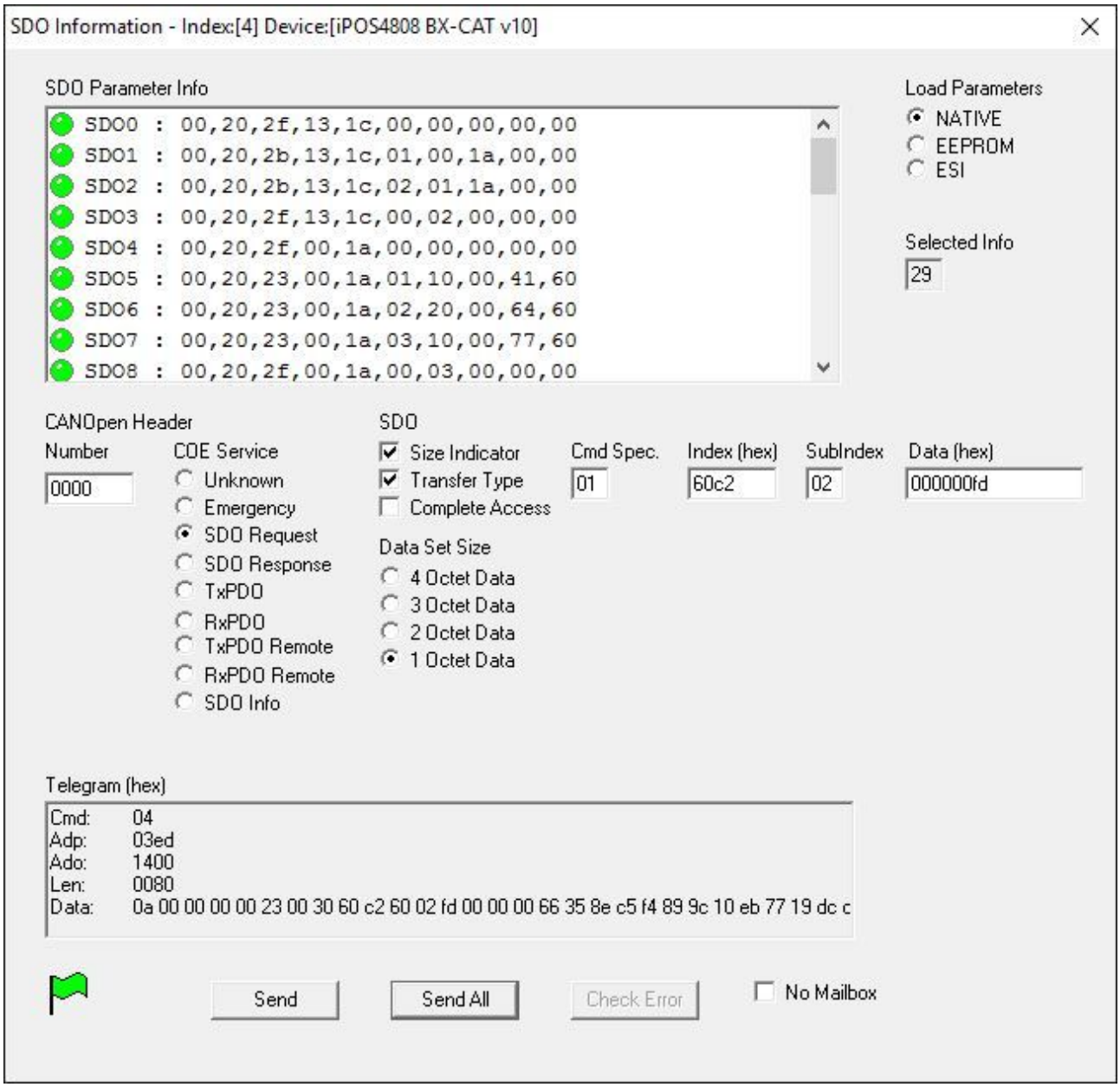
The SYNCMAN configuration dialog allows parsing XML information, EEPROM (SII) information and the Native format for configuration and provides information to all items (also described in the EtherCAT specification).



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.

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.



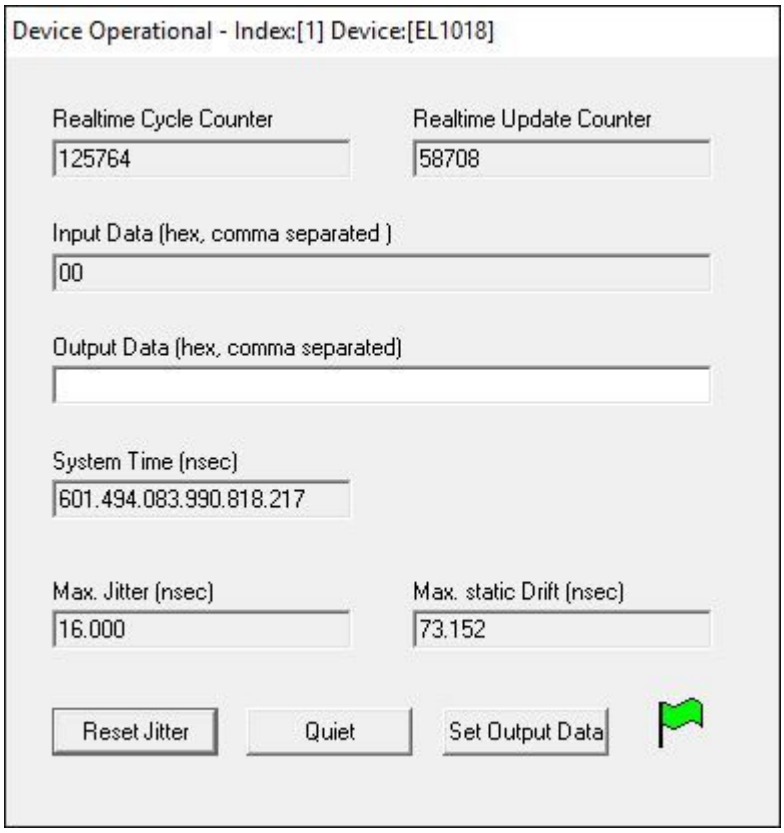
Each SDO information can be selected by “Left Mouse Double Click” on the corresponding line. When the SDO is selected it can be sent to the device. When all SDO information is sent, configuration task is fulfilled

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];
```

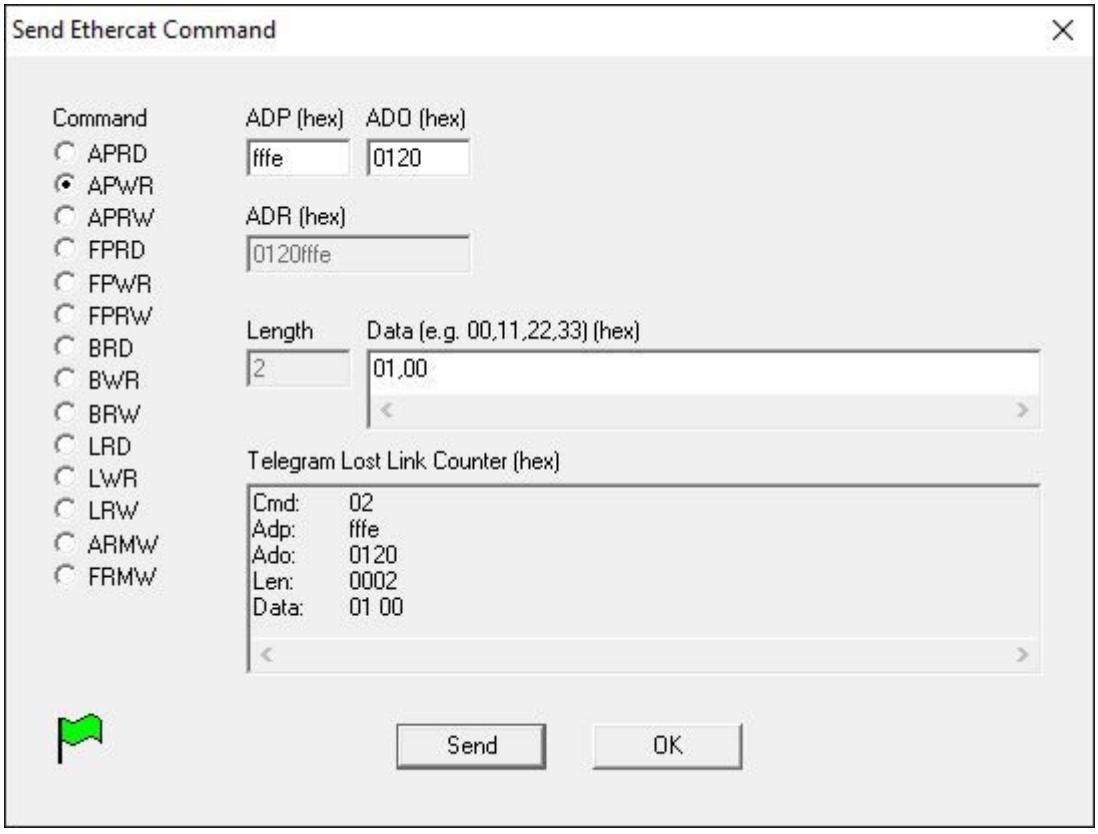
Since many devices support Distributed Clock management, the local system time of the device allows exact jitter and drift measurement.



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.

8.3 Sending EhterCAT Command

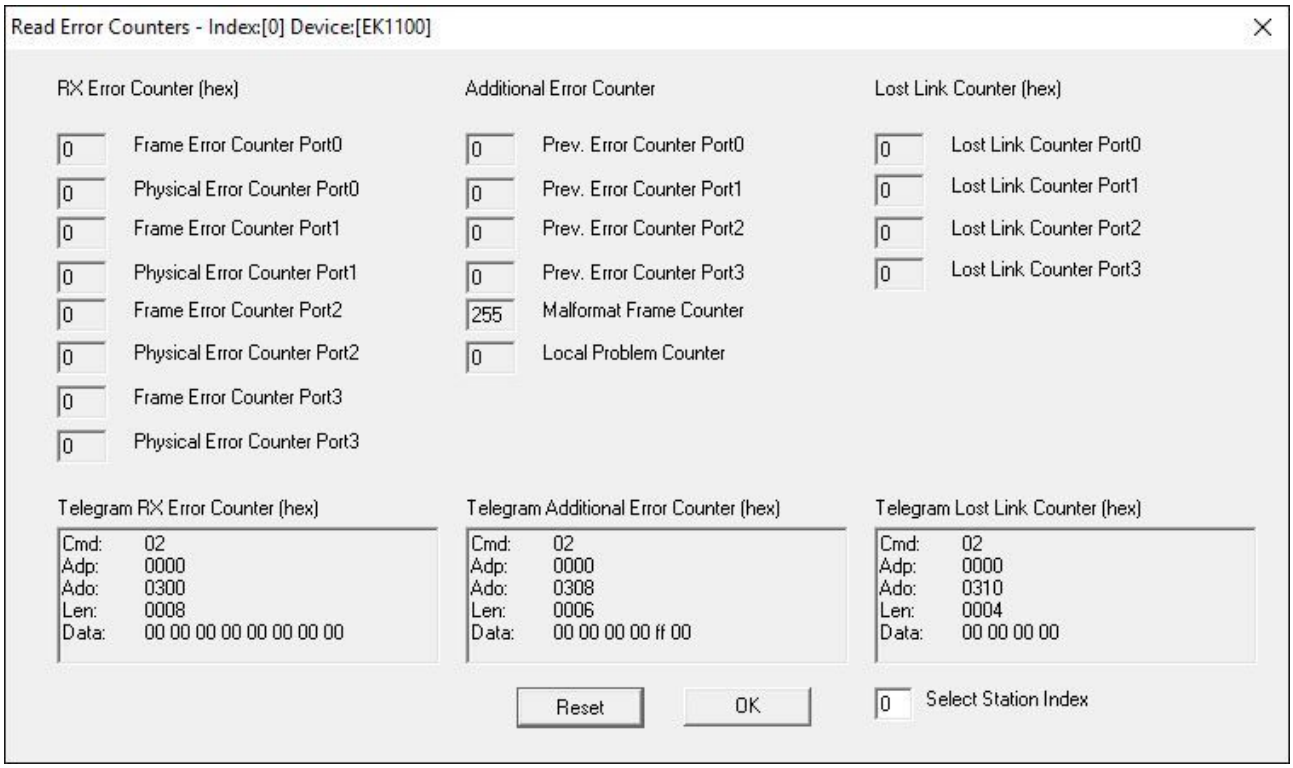
ECATVERIFY allows building and sending of single EtherCAT Commands for test purposes.



8.4 Error Counters

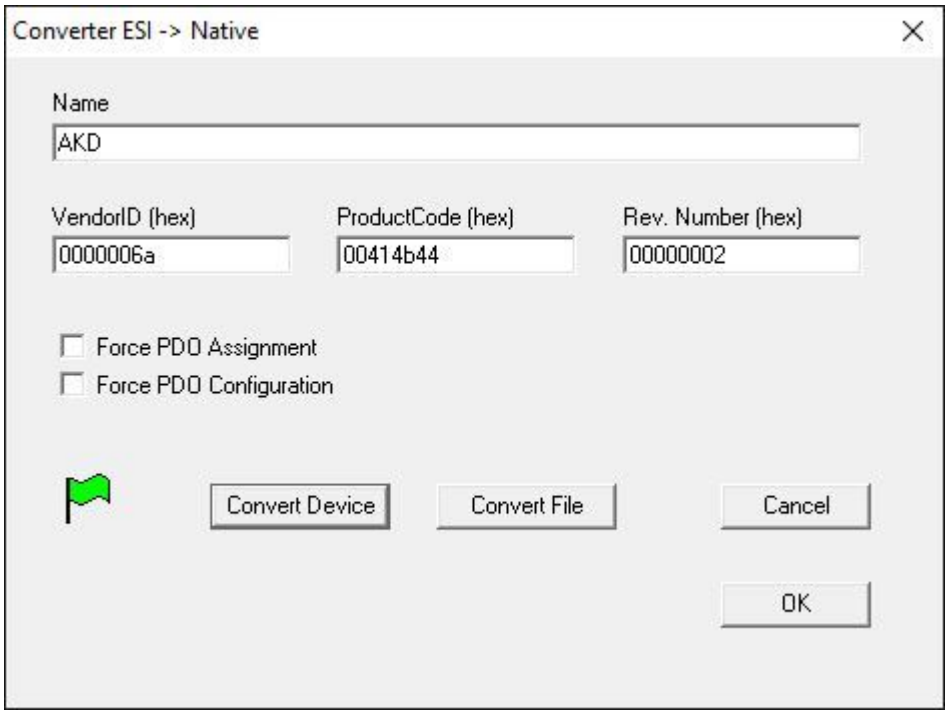
ECATVERIFY gets information about the ErrorCounters

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



8.5 ESI Converter

ECATVERIFY has an implemented XML parser which allows converting XML (ESI) device information into a native format and save it into the parameter file ECATDEVICE[name].PAR (to be placed in the execution directory). Therefor the XML files must be located in the directory where ECATVERIFY resides. 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[name].PAR will be updated.

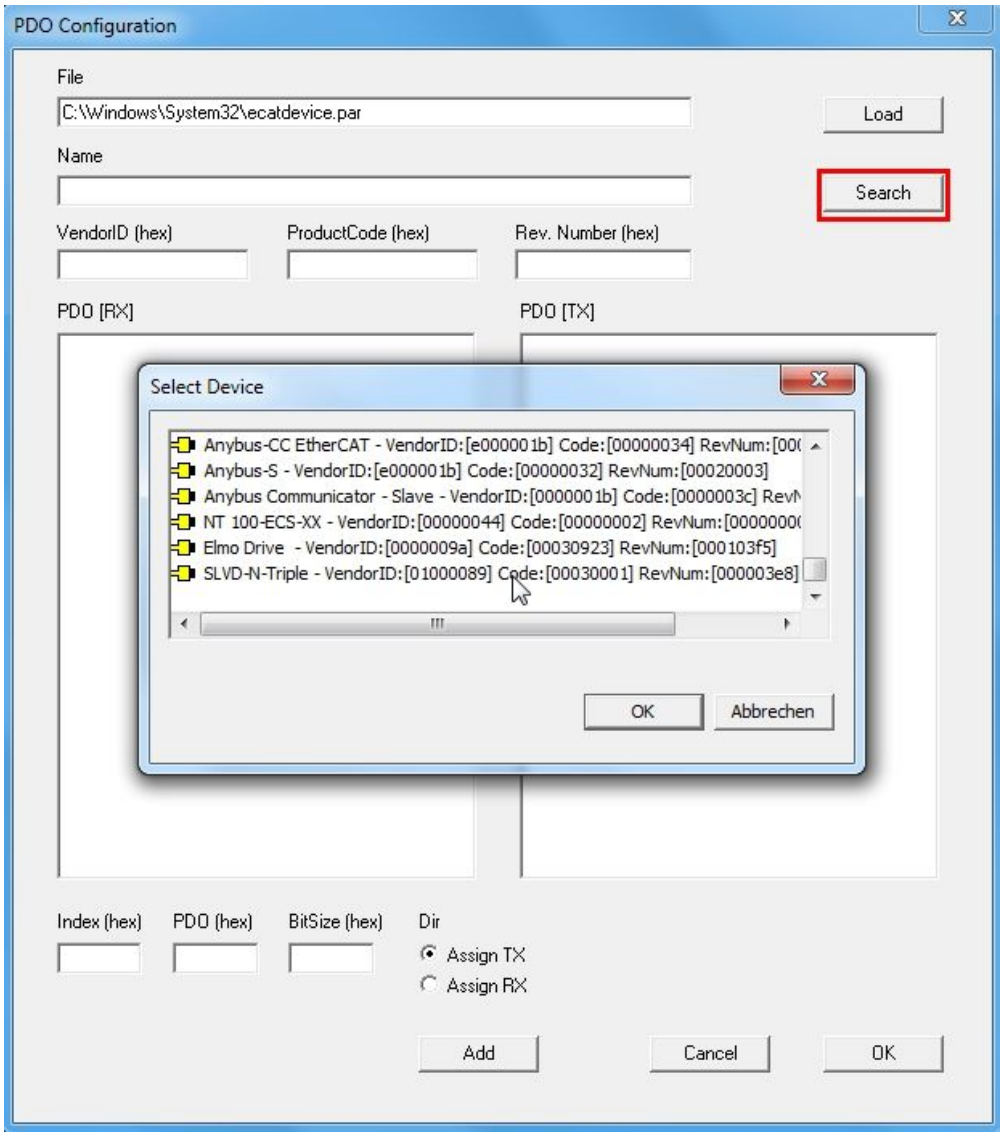


Note: Naming convention of native parameter files

EcatDevice.par
EcatDevice_xxx.par
EcatDevicexxx.par

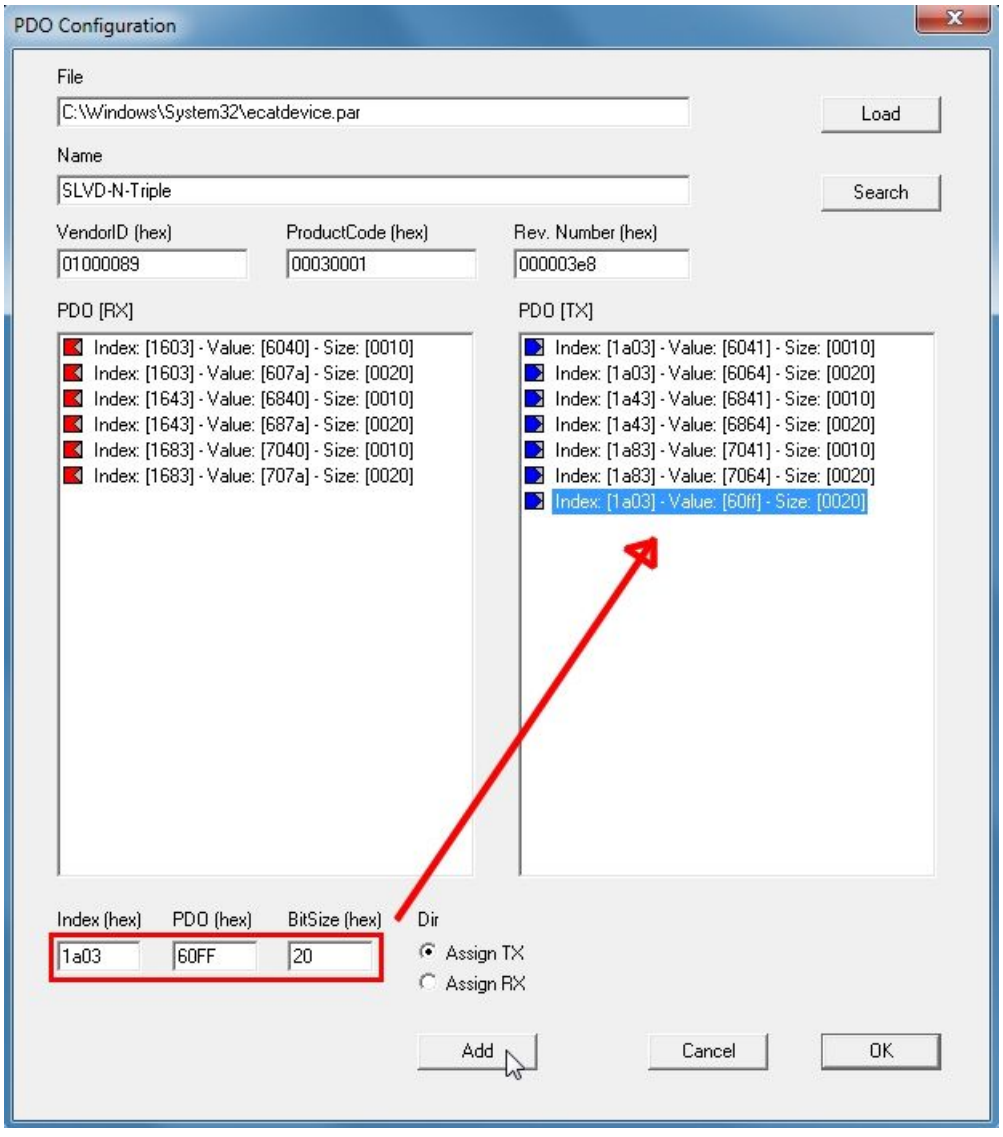
8.6 PDO Configurator

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[name].PAR can be listed or searched for editing the PDO mappings.



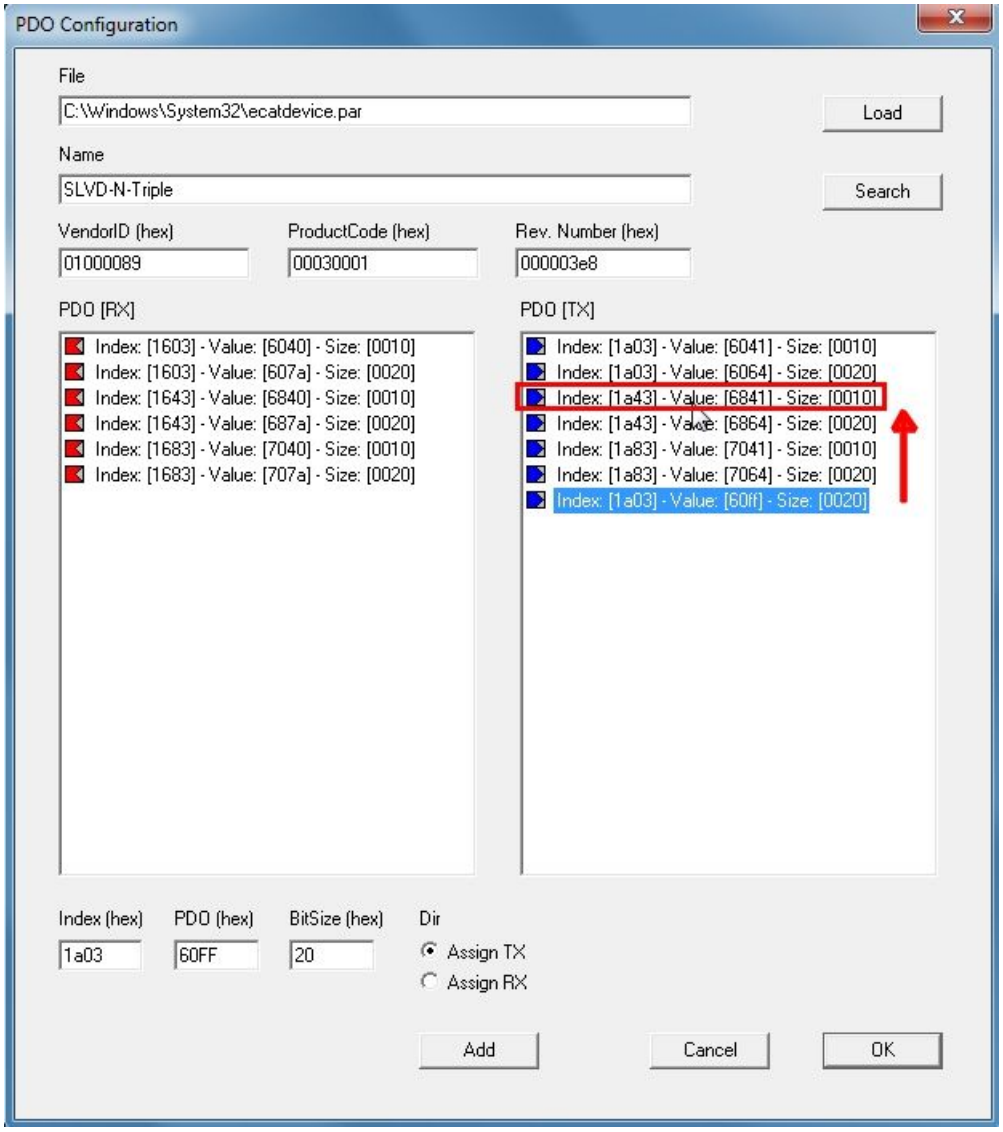
Note: Existing PDO-Mappings need to have an already listed PDO assignment (1C12 / 1C13). Otherwise the PDO mapping has to setup newly.

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

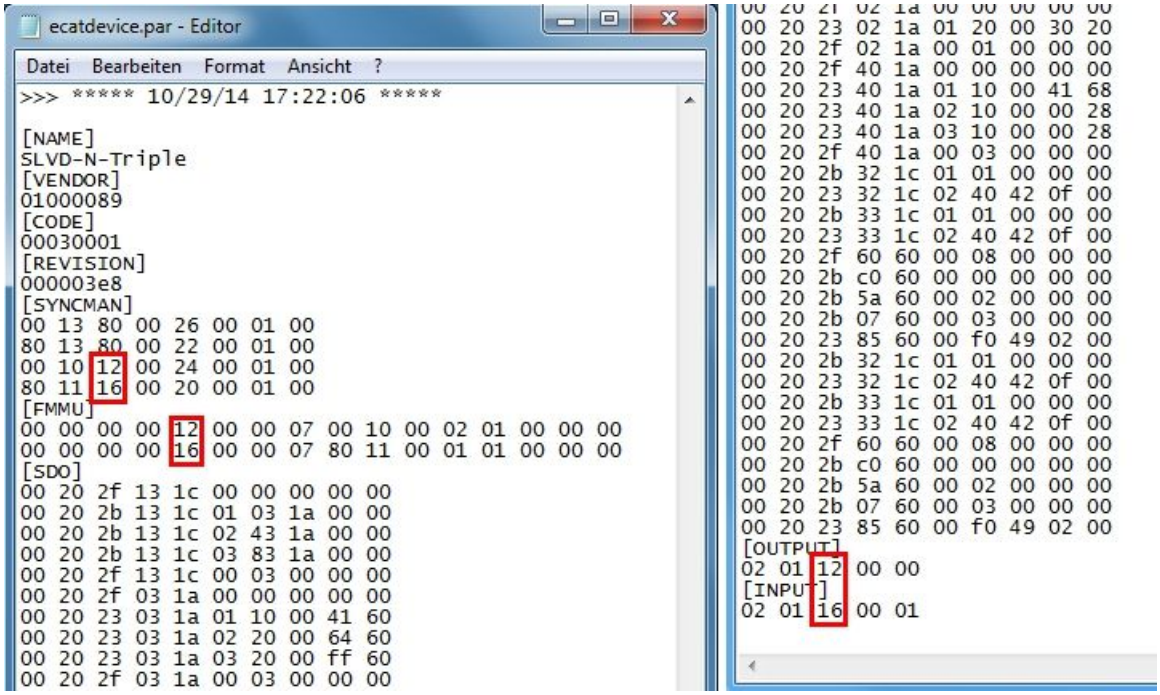


Selected PDO mappings may be deleted by pressing the key „DELETE“.

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.

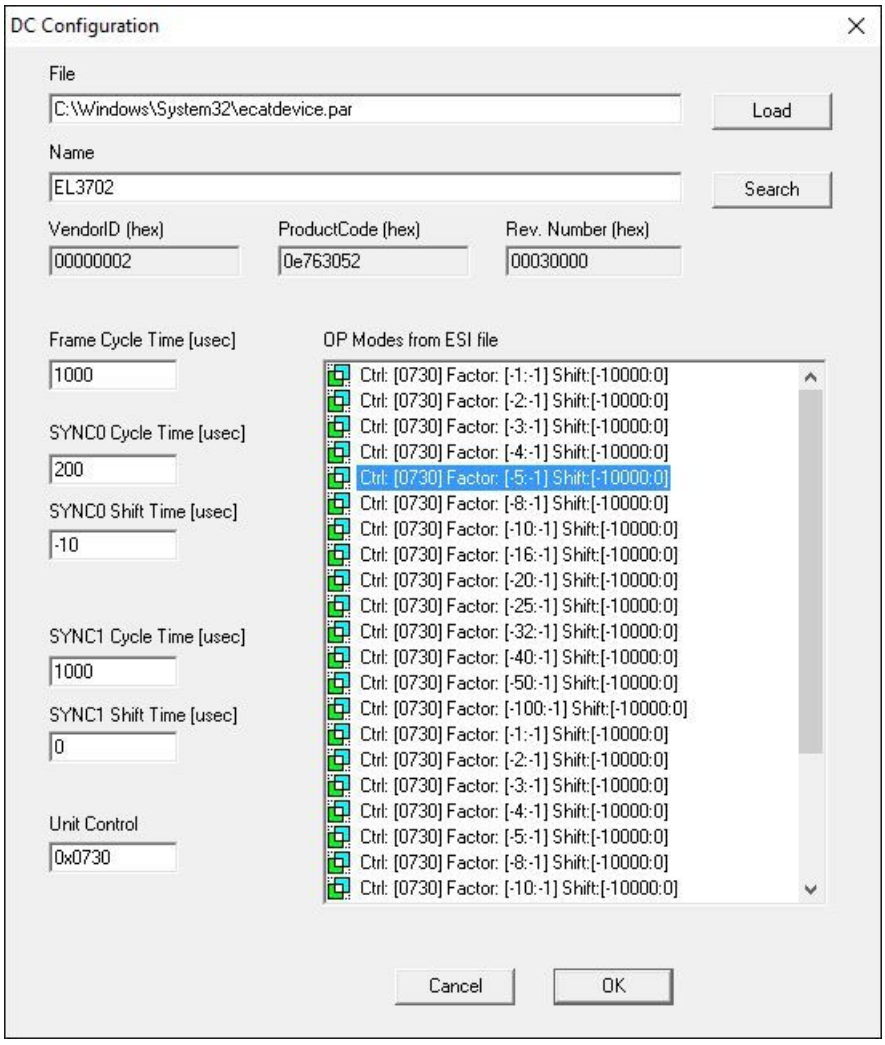


Once configured, the device located in the file ECATDEVICE[name].PAR file is automatically updated and the value “length” of the corresponding FMMU-, SYNCMAN- and INPUT / OUTPUT descriptor entries is automatically updated.



8.7 DC Configurator

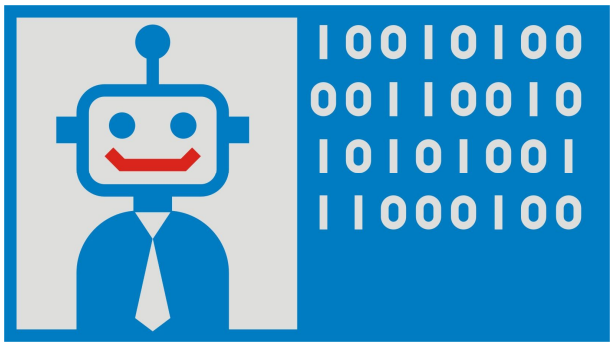
The integrated DC configurator allows easy configuration of distributed clock operation modes. Therefore the ESI file must be placed in the same directory where ECATVERIFY resides.



Result in ECATDEVICE[name].PAR:

```
[OPMODE]
30 07 40 0d 03 00 40 42 0f 00 f0 d8 ff ff 00 00 00 00 ff ff ff ff
```

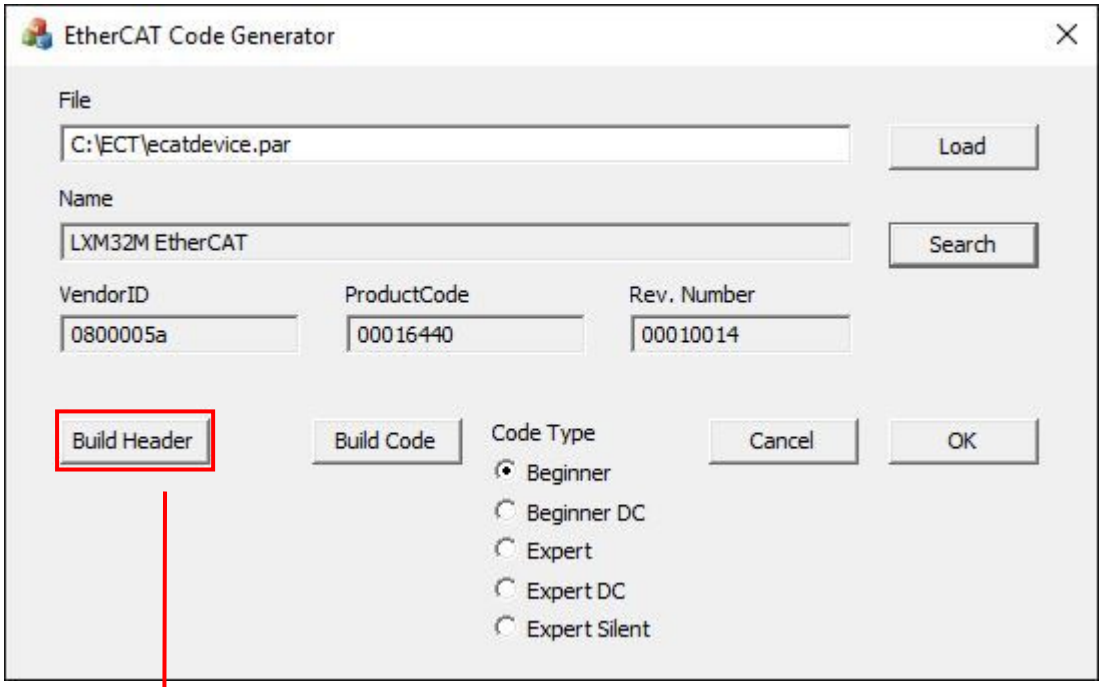
8.8 Code Assistant

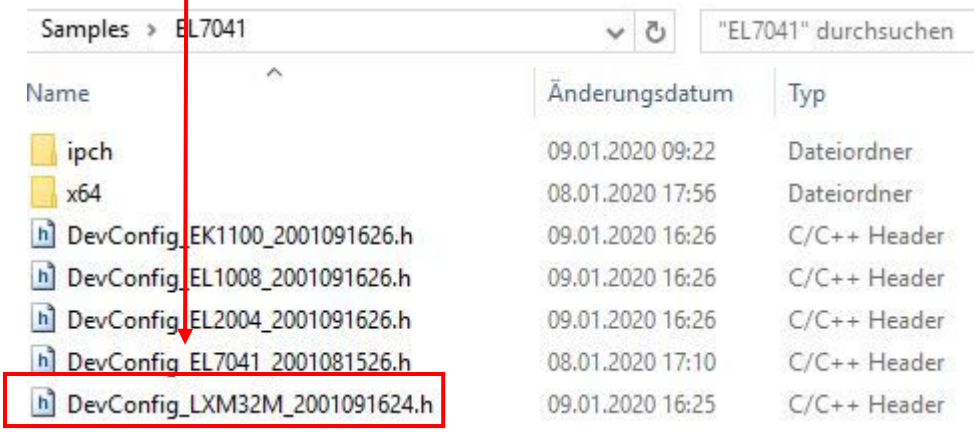


To simplify the handling of the EtherCAT Master library, the code assistant help you to generate a framework code for the EtherCAT communication, based on C++. With the code assistant, a template code will be generated, including header files and code structures. There are 2 steps required: building a header file for each selected device, then building the template code. The code assistant therefor provides several coding levels, from beginner to expert. So the structure of the application code is given, and ready to run. So it makes the handling of the EtherCAT Master Library much easier.

8.8.1 Build Header Files

First, we have to build all header files out of the native parameter file ECATDEVICExxx.par. Therefore we have to select each device by VendorID, ProductCode and Revision. The header file name is represented by a required Prefix (DevConfig_...), the name of the device (e.g. LXM32M) and a date/time stamp (e.g. 2001091624)





Sample: DevConfig_EL7041_2001081526

```
//*****************************************************************
//*** Automated generated header ***
//*** Name:         [EL7041-1000] ***
//*** VendorID:     [0x00000002] ***
//*** ProductCode: [0x1b813052] ***
//*** Revision:     [0x001303e8] ***
//*****************************************************************

#pragma once

//*** Required header files ***
#include <windows.h>
#include "c:\sha\globdef64.h"
#include "c:\sha\Sha64Debug.h"
#include "c:\eth\Sha64EthCore.h"
#include "c:\eth\Eth64CoreDef.h"
#include "c:\eth\Eth64Macros.h"
#include "c:\ect\Sha64EcatCore.h"
#include "c:\ect\Ecat64CoreDef.h"
#include "c:\ect\Ecat64Macros.h"
#include "c:\ect\Ecat64Control.h"
#include "c:\ect\Ecat64SilentDef.h"


//*** Structures need to have 1 byte alignment ***
#pragma pack(push, old_alignment)
#pragma pack(1)

//*** TX mapping structure [Device Output] ***
typedef struct _TX_MAP_EL7041_2001081709
{
      UCHAR  Data0[1];  //Item: Value
      UCHAR  Data1[1];  //Item: Value
      UCHAR  Data2[2];  //Item: Value, Type: Unsigned 16Bit
      UCHAR  Data3[2];  //Item: Value, Type: Unsigned 16Bit
      UCHAR  Data4[2];  //Item: Value

} TX_MAP_EL7041_2001081709, *PTX_MAP_EL7041_2001081709;

//*** RX mapping structure [Device Input]  ***
typedef struct _RX_MAP_EL7041_2001081709
{
      UCHAR  Data0[1];  //Item: Value
      UCHAR  Data1[1];  //Item: Value
      UCHAR  Data2[2];  //Item: Value, Type: Unsigned 16Bit
      UCHAR  Data3[1];  //Item: Value
      UCHAR  Data4[1];  //Item: Value
      UCHAR  Data5[2];  //Item: Value, Type: Signed 16Bit

} RX_MAP_EL7041_2001081709, *PRX_MAP_EL7041_2001081709;

//*** SDO download requests ***
__inline BOOLEAN __SdoControl_EL7041_2001081709(PSTATION_INFO pStation)
{
      ULONG SdoData = 0;
```

```
      SdoData = 0x00000000;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c12, 0x00, 1,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00001600;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c12, 0x01, 2,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00001602;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c12, 0x02, 2,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00001604;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c12, 0x03, 2,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00000003;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c12, 0x00, 1,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00000000;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c13, 0x00, 1,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00001a00;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c13, 0x01, 2,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00001a03;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c13, 0x02, 2,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      SdoData = 0x00000002;
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadReq (pStation, 0x1c13, 0x00, 1,
(PUCHAR)&SdoData))
      if (ERROR_SUCCESS == Ecat64SdoInitDownloadResp(pStation)) {

      //*** Disable SDO automation ***
      pStation->SdoNum = 0;
      return TRUE; }}}}}}}}}

      //*** Something failed ***
      return FALSE;
}

//Set back old alignment
#pragma pack(pop, old_alignment)
```

8.8.2 Build Template Code

Next, we have to build the template code. The code assistant helps you to create several code templates of different type:

Beginner:	Short code, with high level interface. No distributed clock support
Beginner DC:	Short code, with high level interface. Distributed clock support
Expert:	Default code, with low level interface. No distributed clock support
Expert DC:	Default code, with low level interface. Distributed clock support
Expert Silent:	Default code, with low level interface. Distributed clock support and Silent Mode

The code template is being automatically generated with help of already predefined template code file (*.tpl) and the generated header files. There are 3 device specific inserts within the template code:

- Header file includes
- Mapping structures
- SdoControls

This information will be placed inside the template code and makes this code ready to run.

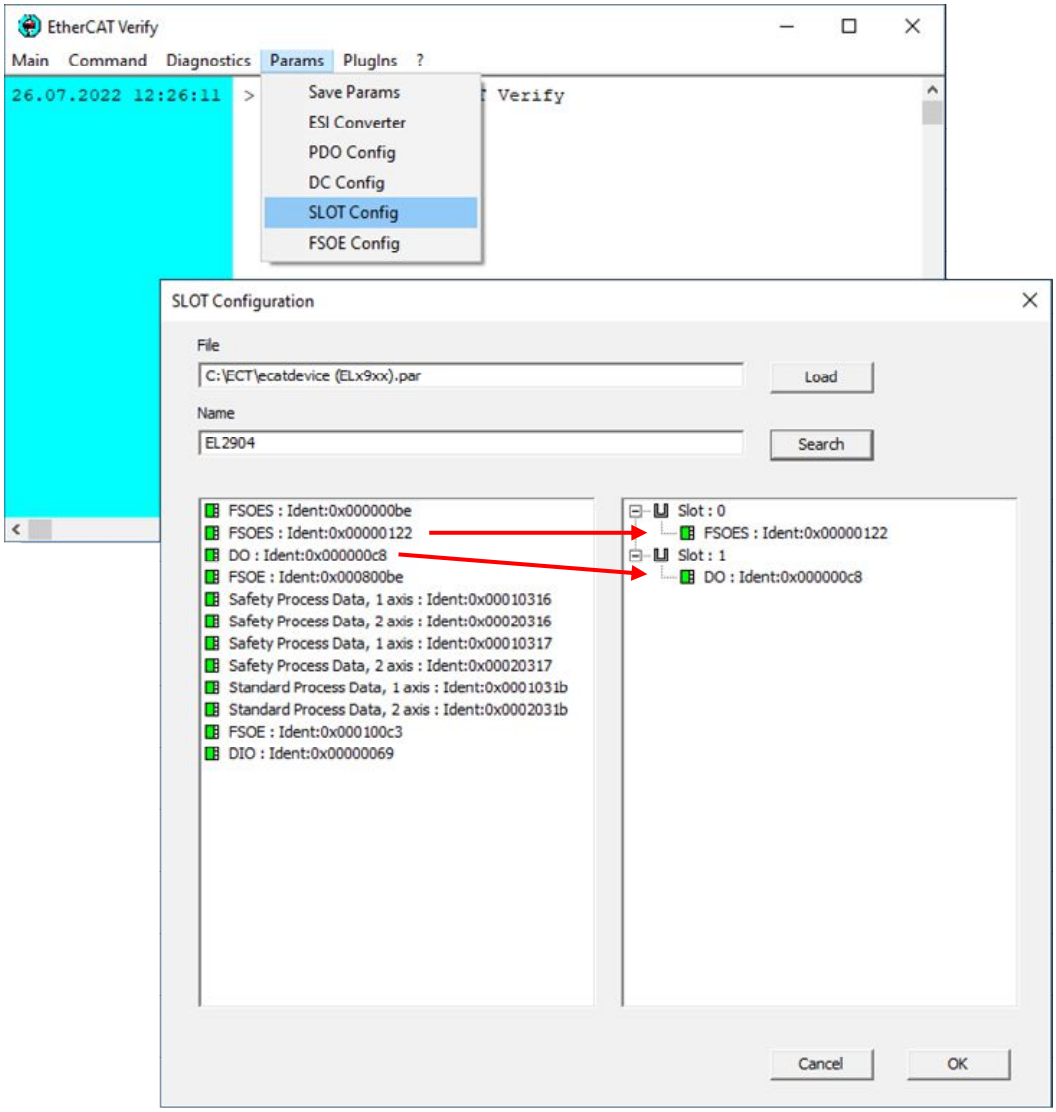
9 FSoE (Fail Safe over EtherCAT)

Important:

The EtherCAT Master Stack of Sybera supports the FSoE protocol.
Although the FSoE protocol is supported, the implemented FSoE stack is not certified !
The user uses the supported FSoE protocol at his own risk !

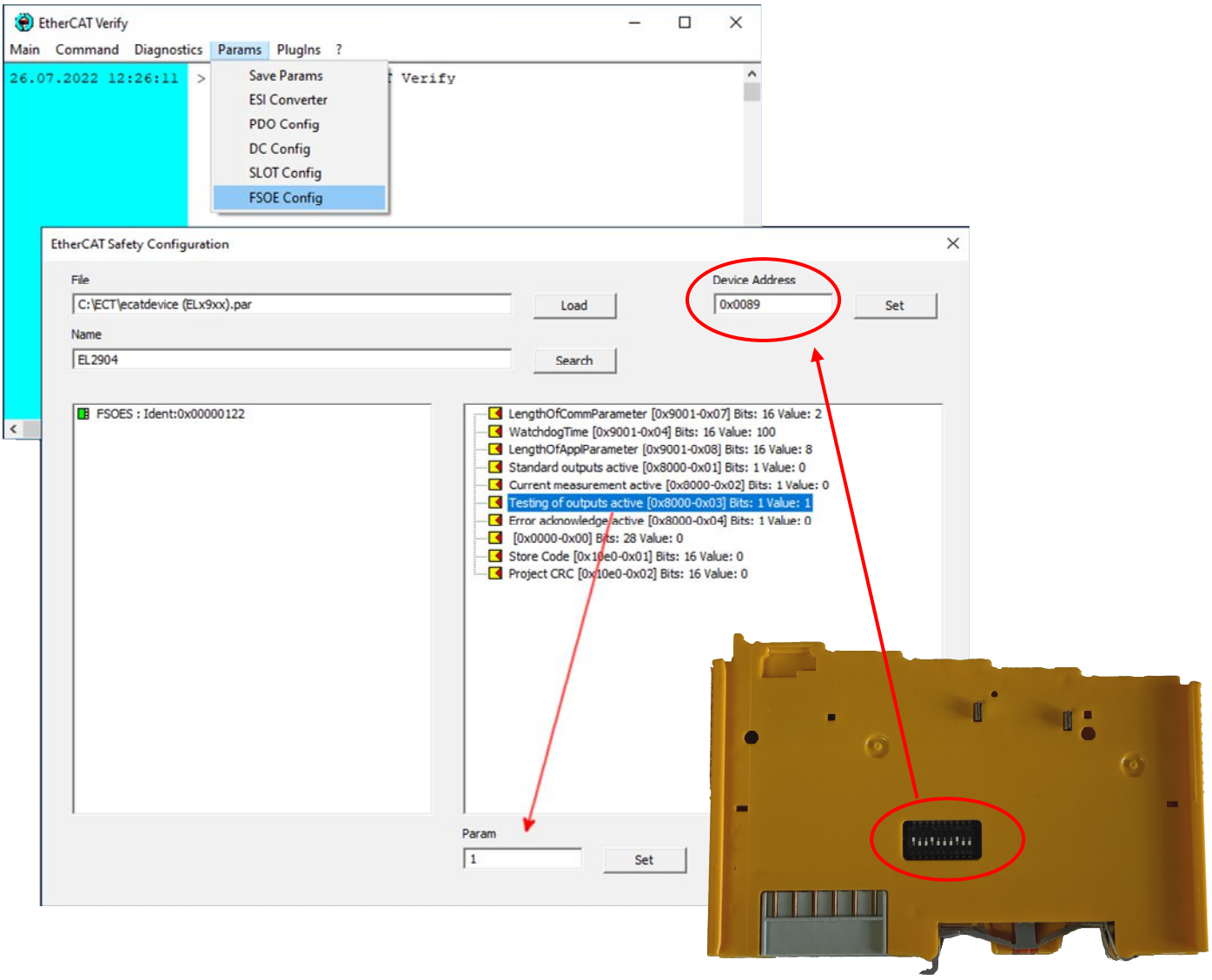
9.1 Slot Configuration

At first, the slots of the FSoE devices have to be assigned. Therefore the slot configuration within the EcatVerify software was designed. Within slots may assigned by drag-and-drop operation.



9.2 FSoE Configuration

Next, the FSoE parameters must be set. Just select the corresponding element, and change the parameter as required. The parameters are described inside the device manual.



The hardware configured address of the FSoE device must be entered as well for a proper configuration.

10 Error Handling

The master library provides several error handling and tracing mechanism.

10.1 Cable Break Detection

Sporadic cable problems within the EtherCAT fieldbus system are often difficult to find. Therefore the EtherCAT master stack and the Ethernet core provide various library functions and elements (for example, error counters) to uncover the problem.

10.1.1 Cyclic Error Detection

The cyclic error detection is typically the first step for finding sporadic bus errors. For this purpose, the event flag IRQ within the RX telegram can be used. This flag is set by the EtherCAT core if the RX and TX working count of the telegram is not different (the working count is incremented by the device during successful command processing in each cycle). Thus, the device position within the network can also be determined where the error occurred. With the flag ERR_FLAG the Ethernet core also provides information, whether an error has occurred during the PHY transmission of the Ethernet adapter.

```
__inline void __CheckError(void)
{
      //Check station error
      for (int i=0; i<__StationNum; i++)
            if (__pSystemList[i].RxTel.s.hdr.irq & (1<<15))
            {
                  //Reset error
                  __pSystemList[i].RxTel.s.hdr.irq &= ~(1<<15);

                  //Set error count and station index
                  __ErrStationIndex = i;
                  __ErrCnt++;
            }

      //Check general PHY error
      __bErrFlag = __pSystemStack->hdr.err_flag;
}
```

10.1.2 Acyclic Error Detection

If an error has been detected in cyclic operation, this can now be verified via an acyclic Ethercat command. For this purpose, the flag ERR_FLAG of the Ethernet core must first be reset, so that the error counter can subsequently be read out.

```
__inline void __CheckStationError(PSTATION_INFO pStation)
{
      RX_ERR_CNT    RxErrCnts = { 0 };
      ADD_ERR_CNT   AddErrCnts = { 0 };
      LOST_LINK_CNT LostLinkCnts = { 0 };

      //First try to reset ethernet core error flag
      if (__pUserStack->hdr.err_flag)
          __pUserStack->hdr.err_flag = FALSE;

      //Do some delay
      Sleep(100);

      //Check flag again
      if (__pUserStack->hdr.err_flag == FALSE)
      {
            //Send ethercat command
            if (ERROR_SUCCESS == Ecat64SendCommand(
                                        FPRD_CMD,
                                        pStation->PhysAddr,
                                        0x300,
                                        sizeof(RX_ERR_CNT),
                                        (PUCHAR)&RxErrCnts))

            if (ERROR_SUCCESS == Ecat64SendCommand(
                                        FPRD_CMD,
                                        pStation->PhysAddr,
                                        0x308,
                                        sizeof(ADD_ERR_CNT),
                                        (PUCHAR)&AddErrCnts))

            if (ERROR_SUCCESS == Ecat64SendCommand(
                                        FPRD_CMD,
                                        pStation->PhysAddr,
                                        0x310,
                                        sizeof(LOST_LINK_CNT),
                                        (PUCHAR)&LostLinkCnts))
            {
                  …
            }
      }
}
```

The error analysis can be further refined, in which the error counters of the Ethernet adapter are called up with the function Sha64EthCheckStatus.

10.2 Debug Log File

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

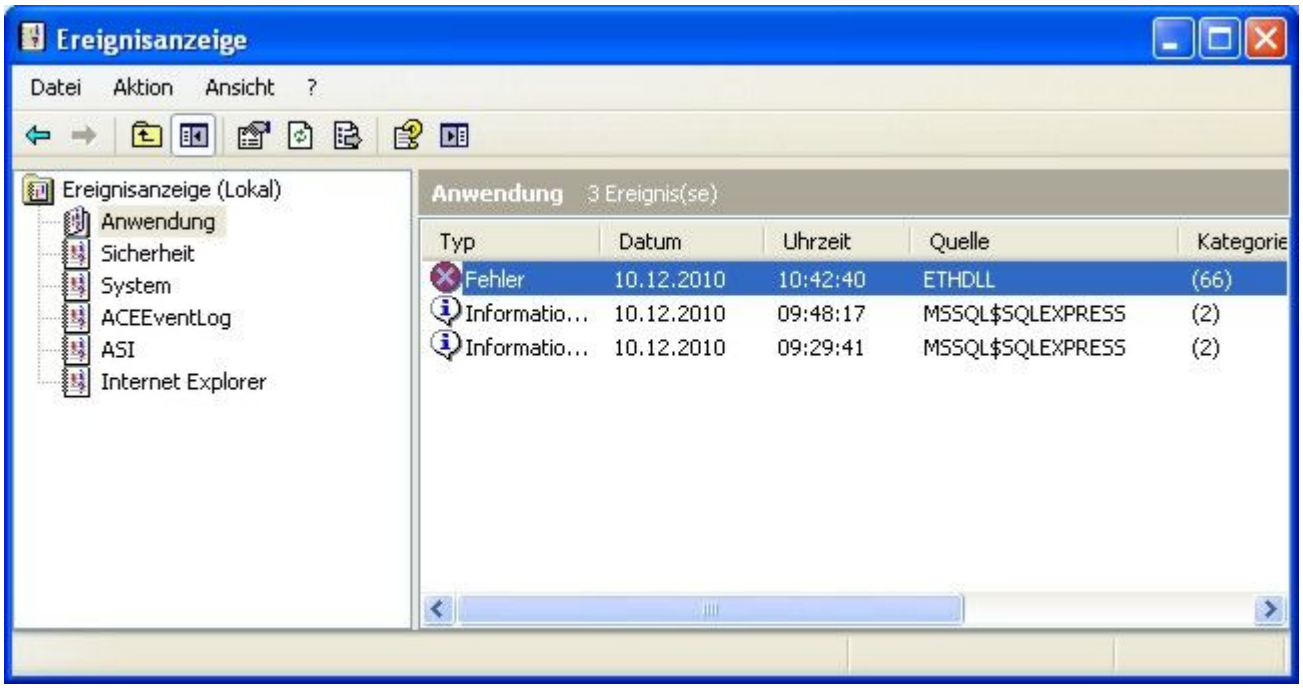
Sample:

```
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
```

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

10.3 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.



11 Related Dokuments

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