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

Date: April, 11.2018
# EtherCAT Realtime Master Library

**Documentation**  
SYBERA Copyright © 2014

## 1 Introduction

- **1.1 Product Features** .......................................................... 5
- **1.2 Supported Platforms** .................................................. 8
- **1.3 Supported OS** ............................................................... 8

## 2 EtherCAT Library Installation

- **2.1 Jitter Control** .............................................................. 9
- **2.2 Dynamic Jitter Compensation** ..................................... 11

## 3 EtherCAT Realtime Master Library

- **3.1.1 Visual Studio 2010 Compiler Settings** ....................... 14
- **3.1.2 Visual Studio 2010 Linker Settings** .......................... 15
- **3.2 Header File ECAT(32/64)COREDEF.H** ....................... 16
- **3.2.1 Structure ECAT_PARAMS** ........................................ 18
- **3.2.2 Structure STATION_INFO** ....................................... 19
- **3.2.3 Structure DATA_DESC** .......................................... 21
- **3.3 Header File ECAT(32/64)MACROS.H** ....................... 22
- **3.4 Header File ECAT(32/64)SDODEF.H** ....................... 23
- **3.5 Header File ECAT(32/64)SIIDEF.H** ............................ 24
- **3.6 Header File ECAT(32/64)DCDEF.H** ....................... 25
- **3.7 Debug Log File** ...................................................... 26

## 4 EtherCAT Library HighLevel Interface

- **4.1.1 Sha(32/64)EcatGetVersion** .................................. 27
- **4.1.2 Sha(32/64)EcatCreate** .......................................... 28
- **4.1.3 Sha(32/64)EcatDestroy** ......................................... 29
- **4.1.4 Sha(32/64)EcatEnable** ........................................... 30
- **4.1.5 Sha(32/64)EcatDisable** ......................................... 31

## 5 Realtime Operation

## 6 EtherCAT Library LowLevel Interface

- **6.1 EtherCAT LowLevel Command Functions** .................. 32
- **6.1.1 Send EtherCAT Command** ..................................... 33
- **6.1.2 Reset Devices** ..................................................... 34
- **6.1.3 Clear Error Counters** ............................................ 35
- **6.1.4 Read DL Information** ........................................... 36
- **6.1.5 Read DL Status** ................................................... 37
- **6.1.6 Read/Write DL Control** ........................................ 38
- **6.1.7 Read PDI Control** ................................................ 39
- **6.1.8 Read PDI Configuration** ....................................... 40
- **6.1.9 Init Station Addresses** ......................................... 41
- **6.1.10 Init Alias Addresses** .......................................... 42
- **6.1.11 Configure SYNC Management** ............................ 43
- **6.1.12 Configure FMMU Management** ........................... 44
- **6.1.13 Configure PDO Assignment** ................................ 45
- **6.1.14 Watchdog Enable** ............................................. 46

- **6.2 EtherCAT LowLevel State Functions** ....................... 47
- **6.2.1 Read AL Status** ................................................... 48
- **6.2.2 Change All States** ............................................... 49
- **6.2.3 Change State By Node Address** ............................ 50
- **6.2.4 Cyclic State Change** ............................................ 51

- **6.3 EtherCAT LowLevel COE Functions** ....................... 52
- **6.3.1 Initiate SDO Download Expedited Request** ........... 53
- **6.3.2 Initiate SDO Download Expedited Response** ......... 54
- **6.3.3 Initiate SDO Upload Expedited Request** ............... 55
- **6.3.4 Initiate SDO Download Expedited Response** ......... 56
6.4 EtherCAT LowLevel Mailbox Functions ....................................................................................49
  6.4.1 Write command to mailbox (sequential) ...........................................................................49
  6.4.2 Read command from mailbox (sequential) ........................................................................49
  6.4.3 Check mailbox for pending response (sequential) ..............................................................49
  6.4.4 Write command to mailbox (parallel) ................................................................................49
  6.4.5 Read command from mailbox (parallel) ............................................................................49
  6.4.6 Check mailbox for pending response ..................................................................................50

6.5 EtherCAT LowLevel EEPROM Functions .............................................................................51
  6.5.1 Read SII Data ......................................................................................................................51
  6.5.2 Write SII Data ....................................................................................................................51
  6.5.3 Reload SII Data ..................................................................................................................51
  6.5.4 Get Category String ..........................................................................................................52
  6.5.5 Get Category String ..........................................................................................................52
  6.5.6 Get Category SYNC Manager .........................................................................................52
  6.5.7 Get Category FMMU Manager ..........................................................................................52
  6.5.8 Get Category PDOs ...........................................................................................................53

6.6 EtherCAT LowLevel Distributed Clock Functions .....................................................................54
  6.6.1 DC Local Time ....................................................................................................................54
  6.6.2 DC Propagation Delay Compensation ..............................................................................54
  6.6.3 DC Offset Compensation ..................................................................................................54
  6.6.4 DC Drift Compensation .....................................................................................................54
  6.6.5 DC Quality Check .............................................................................................................54
  6.6.6 Read DC Synchronisation Information .............................................................................55
  6.6.7 DC Sync Control (indirect) ...............................................................................................55
  6.6.8 DC Sync Control (direct) ..................................................................................................55

7 Device Configuration ....................................................................................................................56
  7.1 Section [NAME] ....................................................................................................................57
  7.2 Section [VENDOR] ................................................................................................................57
  7.3 Section [CODE] ....................................................................................................................57
  7.4 Section [REVISION] ..............................................................................................................57
  7.5 Section [SYNCMAN] .............................................................................................................58
  7.6 Section [FMMU] ....................................................................................................................60
  7.7 Section [SDO] .......................................................................................................................61
    7.7.1 PDO Mapping ..................................................................................................................63
  7.8 Section [OUTPUT] / [INPUT] ...............................................................................................65

8 EtherCAT Verifier (ECATVERIFY) ...................................................................................................66
  8.1 Device List ............................................................................................................................68
  8.2 State Control Dialog ..............................................................................................................69
    8.2.1 Configure Station Address ..............................................................................................71
    8.2.2 Configure FMMU Management .......................................................................................72
    8.2.3 Configure SYNC Management .......................................................................................73
    8.2.4 Configure PDO(s) .............................................................................................................75
    8.2.5 Device Operational .........................................................................................................77

  8.3 Sending EtherCAT Command ..................................................................................................78
  8.4 Error Counters ......................................................................................................................79
  8.5 ESI Converter .........................................................................................................................80
  8.6 PDO Configurator ..................................................................................................................81
  8.7 DC Configurator ....................................................................................................................85

9 Error Handling .............................................................................................................................86
  9.1 Cable Break Detection ..........................................................................................................86
    9.1.1 Cyclic Error Detection .....................................................................................................86
    9.1.2 Acyclic Error Detection ...................................................................................................87
  9.2 Debug LOG File ......................................................................................................................88
9.3 Event File......................................................................................................................................88

10 Related Dokuments................................................................................................................... 89
1 Introduction

The idea of further interface abstraction of the SHA X-Realtime for several communication channels and bus systems, like serial communication, CANBUS, Ethernet (TCP/IP), ... is realized by the SYBERA AddOn Software Modules, so called RealtimeCores. All RealtimeCores are based on the SHA X-Realtime system. The RealtimeCores are intended to fulfill Realtime-Level-1, which means collecting and buffering data in realtime without loss of data, as well as Realtime-Level-2, which means functional operation at realtime. Thus the RealtimeCores usually require simple passive hardware. One of the great benefits is the adjustable scheduling time of incoming and outgoing data.
The EtherCAT realtime library system of SYBERA enables a custom ethernet adapter under Windows as an EtherCAT Master. Therefore the base is the Sybera X-Realtime technology. The library system allows the deterministic control of EtherCAT slave participants (e.g., the EtherCAT modules from Beckhoff Automation GmbH). Depending on the PC hardware telegram update cycles upto 50 microseconds are realistic. As physical link customary INTEL or REALTEK chips are suitable.

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

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

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

1.2 Supported Platforms
SHA was build to support several development platforms. Currently following platforms are supported:

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

1.3 Supported OS

- Windows VISTA, 7, 8, 10 (32 / 64 Bit)
2 EtherCAT Library Installation

For installation following steps are required:

Preparation

1. Provide a PC with INTEL or REALTEK Ethernet adapter and Windows operating system (with administrator privileges)
2. Make sure, the latest Updates are installed (especially Windows 7)
3. Install SHA realtime system (separate software package)
4. Install ETH transport library (separate software package)

Installation

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

   On Installation the PEC information (PID, SERNUM and KEYCODE) must be entered. The PID for the evaluation version is 11223344, the SERNUM is 11223344, the KeyCode therefore is: 00001111-22223333

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

Operation

7. Run ECATVERIFY(32/64).EXE (with administrator privileges)
8. Build device description ECATDEVICE.PAR
   (must be placed in C:\WINDOWS\SYSTEM32)
9. Build the program with the library interface
10. Run the program

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

Note: In order to operate SYBERA software under Windows 8, 7, VISTA, it must be carried out with ADMINISTRATOR privileges.
Note: For proper operation, make shure within the BIOS the INTEL Speedstep Technologie, the INTEL TurboBoost Technologie as well as the INTEL C-STATE Technologie is turned off.

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

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

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

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

Following values are valid:

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

SYBERA uses the procedure "Dynamic Jitter Compensation" with active and passive feedback compensation within the realtime engine. Although the X-Real time engine of SYBERA allows a native maximum Jitter of approx. 15 µ sec (according to hardware platform), this behaviour may be reduced below 3 µsec by the dynamic jitter compensation.

For compatibility reason on some platforms it may be required to disable the dynamic jitter compensation. Therefore the registry value “NoJitterComp” has to be set to 1.
Note: For proper operation it is recommended to use the EtherCAT network as a standalone network. This requires turning off the Windows protocols for this network connection:
3 EtherCAT Realtime Master Library

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

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

Sample Application

![EtherCAT Core Realtime Level 2 Test](image)
3.1.1 Visual Studio 2010 Compiler Settings

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

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

![Visual Studio 2010 Linker Settings](image)
Sample Startup Protocol:
3.2 Header File ECAT(32/64)COREDEF.H

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

3.2.1 Structure ECAT_PARAMS

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

```c
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
    PSTATION_INFO pUserList; //Station List Pointer
} ECAT_PARAMS, *PECAT_PARAMS;
```

Note:

The structure ETH_PARAMS is part of the Ethernet Core Library and described in 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
    DC_LOCAL_TIME DcLocalTime; //DC Local Time
    DC_SYNC_INFO DcSyncInfo; //DC Sync Information
    RX_ERR_CNT RxErrCnt; //RX Error Counter
    FMMU FmmuList[MAX_FMMU_NUM]; //FMMU Manager List
    ULONG FmmuNum; //Number of FMMU records
    SYNCMAN SyncManList[MAX_SYNCMAN_NUM]; //SYNCMAN Manager List
    ULONG SyncManNum; //Number of SYNCMAN records
    SDO_LEGACY SdoList[MAX_SDO_NUM]; //SDO Legacy Command List
    ULONG SdoNum; //Number of SDO commands
    USHORT PhysAddr; //Physical Station Address
    USHORT AliasAddr; //Alias Station Address
    ECAT_TELEGRAM TxTel; //TX Process Telegram
    ECAT_TELEGRAM RxTel; //RX Process Telegram
    DATA_DESC OutDescList[MAX_DATA_DESC]; //Output Descriptor List
    ULONG OutDescNum; //Number of TX Data Descriptors
    DATA_DESC InDescList[MAX_DATA_DESC]; //Input Descriptor List
    ULONG InDescNum; //Number of RX Data Descriptors
    BOOLEAN bUpdate; //Station Update Flag (read only)
    BOOLEAN bDisable; //Station Disable Flag
    UCHAR Reserved[MAX_RESERVED_SIZE]; //Reserved Data Size
} STATION_INFO, *PSTATION_INFO;
Note:

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

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

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

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

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

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

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

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

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

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

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

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

    //Set command, address and len due to descriptors
    if ((pStation->OutDescNum != 0) && (pStation->InDescNum != 0))
```
{ Cmd = LRW_CMD; DataSize = DataSize; Addr.bit32 = LogicalAddr; }

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

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

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

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

//Set station update
pStation->bUpdate = TRUE;
3.3  Header File ECAT(32/64)MACROS.H

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

This Inline-Macro is to set telegram information:

```c
__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:

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

This Inline-Macro is to get telegram information:

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

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

This Inline-Macro is to copy telegrams:

```c
__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(32/64)SDODEF.H

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

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

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

//*** SDO Legacy Request

typedef union _SDO_LEGACY
{
    UCHAR bytes[1];
    struct
    {
        COE_HDR CoeHdr;
        SDO_INIT_HDR SdoHdr;
        TYPE32 Data;
    } s;
} SDO_LEGACY, *PSDO_LEGACY;
3.5 Header File ECAT(32/64)SIIDEF.H

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

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

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

The EtherCAT master library provides a buildin log sytem which produces a debug log file called \textit{ECATDBG.LOG}. This file contains all necesary information of the library sequence.

\textbf{Sample:}

\begin{verbatim}
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
\end{verbatim}
4 EtherCAT Library HighLevel Interface

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

VC : Visual C and Borland C++ Builder
VB : Visual Basic
DP : Borland Delphi

4.1.1 Sha(32/64)EcatGetVersion

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

VC

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

Sample:

```c
//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(32/64)EcatCreate

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

```c
VC ULONG Sha(32/64)EcatCreate (PECAT_PARAMS);
```

Sample:

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

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

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

```
EcatParams.EthParams.period * EcatParams.SyncCycles
```

(e.g. 100µsec * 20 = 2msec)
Note: Library LowLevel Routines implemented by ShaEcatCreate

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

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

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

Note: Logical Addressing Scheme

The EtherCAT Realtime Library provides an integrated logical addressing scheme. Thereby all EtherCAT stations get a 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);
}
4.1.3 Sha(32/64)EcatDestroy
This function closes the EtherCAT communication.

VC ULONG Sha(32/64)EcatDestroy(PECAT_PARAMS);

4.1.4 Sha(32/64)EcatEnable
This function enables the EtherCAT station list and must follow the function ShaEcatCreate.

VC ULONG Sha(32/64)EcatEnable(PECAT_PARAMS);

**Note:** Library LowLevel Routines implemented by this Function

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

4.1.5 Sha(32/64)EcatDisable
This function disables the EtherCAT station list

VC ULONG Sha(32/64)EcatDisable(PECAT_PARAMS);
Sample:

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

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

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

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

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

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

    //Check ready count for cycle operation
    if (__ReadyCnt == 1)
    {
        //*********************************
        //Do the logical station operation
        //e.g. toogle output value (station index == 2)
        __pSystemList[2].TxTel.s.data[0] =
            (__pSystemList[1].TxTel.s.data[2]) ? 0x00 : 0x03;
    }
void main(void)
{
    printf("\n*** EtherCAT Core Realtime Level2 Test ***\n\n");

    //Required ECAT parameters
    ECAT_PARAMS EcatParams;
    EcatParams.PhysAddr = DEFAULT_PHYSICAL_ADDRESS;
    EcatParams.LogicalAddr = DEFAULT_LOGICAL_ADDRESS;
    EcatParams.SyncCycles = 10;
    EcatParams.EthParams.dev_num = 0;
    EcatParams.EthParams.eth_type = ETH_TYPE_ECAT; //Set ethernet frame type
    EcatParams.EthParams.eth_if = ETH_IF_CORE; //Set filter interface
    EcatParams.EthParams.period = 200; //Set realtime period
    EcatParams.EthParams.sched_cnt = 1; //Set application task
    EcatParams.EthParams.fpAppTask = AppTask;

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

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

            //Call exit function
            __fpEcatExit();

            //Increase loop count
            __LoopCnt++;
printf("\nRemain Time: %i\n\n",
EcatParams.EthParams.remain_time);

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

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

    // Do some delay
    Sleep(100);
}

// Disable Stations
ShaEcatDisable(&EcatParams);

// Destroy ECAT core
ShaEcatDestroy(&EcatParams);
5 Realtime Operation

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

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

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

```c
//Define ECAT states
definition _ECAT_STATE
{
    ECAT_STATE_INIT = 0,  //Initial state
    ECAT_STATE_UPDATE,    //Update still in progress
    ECAT_STATE_READY,     //All stations are updated
    ECAT_STATE_ERROR      //An update error occured
};
```
The Wrapper Functions require as parameters the Ethernet Stack pointer (e.g. __pSystemStack), the station list pointer (e.g. __pSystemList) and the number of stations (e.g. __StationNum). These parameters and others are returned when initializing the EtherCAT Realtime Library and are set as global elements.

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

![Diagram showing EtherCAT state transitions and sampling periods]

**Sampling Period = 100 µsec**

**SyncCycles = 10**
void static AppTask(void)
{
    // Check if system memory is present
    if ((__pSystemStack) ||
        (!__pSystemList))
        return;

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

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

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

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

        __UpdateCnt++;
    }

    // Call exit function
    __fpEcatExit();

    // Increase loop count
    __LoopCnt++;
}
6 EtherCAT Library LowLevel Interface

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

6.1 EtherCAT LowLevel Command Functions

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

6.1.1 Send EtherCAT Command

This is an universal function for sending EtherCAT commands

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

Sample:

```
//Send ethercat command
ULONG Result = EcatSendCommand(APWR_CMD, 0xFFFE, 0x120, 2, (PUCHAR)"\01\00");
```

6.1.2 Reset Devices

This command proceeds following actions:

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

```
ULONG Result = Ecat(32/64)ResetDevices(void);
```
6.1.3 Clear Error Counters
Read or Reset RX Error Counter: BWR Offs 0x300 - 0x303

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

6.1.4 Read DL Information
Read DL information into station list

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

6.1.5 Read DL Status
Read DL Status information into station list

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

6.1.6 Read/Write DL Control
Read/Write DL Control information into station list

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

6.1.7 Read PDI Control
Read PDI Control information into station list

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

6.1.8 Read PDI Configuration
Read PDI Configuration information into station list

ULONG Result = Ecat(32/64)ReadPDIConfig(void);
6.1.9 Init Station Addresses
Initialize all station physical addresses, beginning from the physical start address

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

6.1.10 Init Alias Addresses
Initialize all station alias addresses (requires station element AliasAddr set before)

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

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

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

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

ULONG Result = Ecat(32/64)InitFmmus(void);
6.1.13 Configure PDO Assignment

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

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

This command proceeds following actions:
- Check mailbox for pending response
- Write COE command to mailbox
- Read COE command from mailbox
- Check SDO response

6.1.14 Watchdog Enable

Enables/Disables all watchdog controls of the station list

```
ULONG ECatWatchdogEnable(BOOLEAN bEnable)
```
6.2 EtherCAT LowLevel State Functions

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

6.2.1 Read AL Status

Read AL status of all stations: APRD Offs 0x130

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

6.2.2 Change All States

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

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

6.2.3 Change State By Node Address

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

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

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

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

For DC purpose it’s recommended to use 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 PRE OPERATIONAL
    // Init PDO assignment
    if (ERROR_SUCCESS == Ecat64ChangeAllStates(AL_STATE_PRE_OP))
        if (ERROR_SUCCESS == Ecat64PdoAssignment())
            {
            // Drift compensation delay [msec]
            ULONG CompDelay = 5000;

            // 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(&CompDelay))
                            if (ERROR_SUCCESS == Ecat64DcControl())
                                {
                                // Init process telegrams
                                InitProcessTelegrams ();
```
//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);  
        ...
    }
6.3 EtherCAT LowLevel COE Functions

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

6.3.1 Initiate SDO Download Expedited Request

This function initiates a SDO Download Expedited Request

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

6.3.2 Initiate SDO Download Expedited Response

This function initiates a SDO Download Expedited Response

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

6.3.3 Initiate SDO Upload Expedited Request

This function initiates a SDO Upload Expedited Request

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

6.3.4 Initiate SDO Download Expedited Response

This function initiates a SDO Download Expedited Response

```c
ULONG Ecat(32/64)SdoInitUploadResp( 
PSTATION_INFO pStation, 
PULONG pSdoDataSize, 
P UCHAR* ppSdoData)
```
Sample:

```c
//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);
```
6.4 EtherCAT LowLevel Mailbox Functions

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

6.4.1 Write command to mailbox (sequential)

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

6.4.2 Read command from mailbox (sequential)

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

6.4.3 Check mailbox for pending response (sequential)

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

6.4.4 Write command to mailbox (parallel)

```
ULONG Result = Ecat(32/64)MailboxWriteAll(
    PMAILBOX_INFO pInfoList,
   ULONG InfoNum,
   UCHAR MailboxType)
```

6.4.5 Read command from mailbox (parallel)

```
ULONG Result = Ecat(32/64)MailboxReadAll( 
    PMAILBOX_INFO pInfoList,
   ULONG InfoNum)
```
6.4.6 Check mailbox for pending response
This function checks a mailbox for pending response

```c
ULONG Result = Ecat(32/64)MailboxCheckAll(
    PMAILBOX_INFO pInfoList,
    ULONG InfoNum)
```
6.5 EtherCAT LowLevel EEPROM Functions

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

6.5.1 Read SII Data

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

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

6.5.2 Write SII Data

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

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

6.5.3 Reload SII Data

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

```c
ULONG Result = Ecat(32/64)SiiReload(  
PSTATION_INFO pStation,  
USHORT DataSize,  
USHORT Offs)
```
6.5.4 Get Category String
This function searches inside the SII area for a general information due to a given index.

U_LONG SHAAPI Ecat(32/64)GetCategoryGeneral(
    UCHAR pCatArea,
    U_LONG CatAreaSize,
    UCHAR pGeneral)

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

U_LONG Result = Ecat(32/64)GetCategoryString(
    UCHAR pCatArea,
    U_LONG CatAreaSize,
    char* pszStr,
    U_LONG StrIndex)

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

U_LONG Result = Ecat(32/64)GetCategorySyncman(
    UCHAR pCatArea,
    U_LONG CatAreaSize,
    UCHAR pSyncman,
    U_LONG SyncmanIndex)

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

U_LONG Result = Ecat(32/64)GetCategoryFmmu(
    UCHAR pCatArea,
    U_LONG CatAreaSize,
    UCHAR pFmmu,
    U_LONG FmmuIndex)
6.5.8 Get Category PDOs

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

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

Sample:

```c
//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);
```
6.6 EtherCAT LowLevel Distributed Clock Functions

The EtherCAT realtime library provides functions for propagation delay compensation, system time offset compensation and drift compensation. Additionally 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 to PRE-OPERATIONAL state.

6.6.1 DC Local Time
This function latches out the local time of all stations.

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

6.6.2 DC Propagation Delay Compensation
This function compensates the propagation delay for the stations relations

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

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

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

6.6.4 DC Drift Compensation
This function compensates the drift of the DC station clock by writing ARMW commands at least 3 seconds.

ULONG Result = Ecat(32/64)CompDcDrift(ULONG InitTime);

6.6.5 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 = Ecat(32/64)CheckDcQuality(PULONG pMaxSysTimeDiff);
6.6.6 Read DC Synchronisation Information
This function reads the complete DC_SYNC_INFO structure for further DC processing

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

6.6.7 DC Sync Control (indirect)
This function enables the synchronisation output signal, due to the DC configuration in ECATDEVICE.PAR.

ULONG Result = Ecat64DcControl(VOID);

6.6.8 DC Sync Control (direct)
This function enables the synchronisation output signal, due to the DC settings.

ULONG SHAAPI Ecat(32/64)SyncControl(
    PSTATION_INFO pStation,
    ULONG Sync0CycleTime,
    ULONG Sync1CycleTime,
    ULONG Sync0CycleShift,
    ULONG Sync1CycleShift,
    BOOLEAN bSync0Pulse,
    BOOLEAN bSync1Pulse,
    BOOLEAN bSyncPdiCtrl)

Sample:

Ecat(32/64)SyncControl(
    &__pUserList[i],
    Period * SyncCycles * 1000, //Sync0 cycle time [nsec]
    0, //Sync0 cycle time [nsec]
    20*1000, //Sync0 cycle shift [nsec]
    0, //Sync0 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.
7 Device Configuration

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

Sample:

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

[NAME]
EL3102
[VENDOR]
00000002
[CODE]
0c1e3052
[REVISION]
00100000
[SYNCM]
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
Note: With newer devices the configuration is stored inside the EEPROM. The EtherCAT Master Library is able to configure the devices by parsing the EEPROM information, even without XML file or Native file. But without using the configuration file, the configuration time increases by parsing EEPROM information. The Software ECATVERIFY parses XML information and EEPROM information and converts it into the native format and gives additional help for configuration.

7.1 Section [NAME]
This section contains the name of the device:

(NAME)

EL3102

7.2 Section [VENDOR]
This section contains the vendor ID of the device:

(VENDOR)

00000002

7.3 Section [CODE]
This section contains the product code of the device:

(CODE)

0C1E3052

7.4 Section [REVISION]
This section contains the revision number of the device:

(CODE)

00100000
7.5 Section [SYNCMAN]

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

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

Meaning:

```
<table>
<thead>
<tr>
<th>Addr</th>
<th>Len</th>
<th>Cntr</th>
<th>ChEn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stat</td>
<td>Res</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
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
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>relative address (offset)</th>
<th>Data type</th>
<th>Access type</th>
<th>Access type PDI</th>
<th>Value/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical start address</td>
<td>0x0000</td>
<td>WORD</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>0x0002</td>
<td>WORD</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Buffer type</td>
<td>0x0004</td>
<td>Unsigned2</td>
<td>RW</td>
<td>R</td>
<td>0x00: buffered</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x02: mailbox</td>
</tr>
<tr>
<td>Direction</td>
<td>0x0004</td>
<td>Unsigned2</td>
<td>RW</td>
<td>R</td>
<td>0x00: area shall be read from the master</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: area shall be written by the master</td>
</tr>
<tr>
<td>reserved</td>
<td>0x0004</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R</td>
<td>0x00</td>
</tr>
<tr>
<td>DLS-user event enable</td>
<td>0x0004</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R</td>
<td>0x00: DLS-user event is not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: DLS-user event is active (when area was accessed and is no longer locked)</td>
</tr>
<tr>
<td>Watchdog enable</td>
<td>0x0004</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R</td>
<td>0x00: watchdog disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: watchdog enabled</td>
</tr>
<tr>
<td>reserved</td>
<td>0x0004</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R</td>
<td>0x00</td>
</tr>
<tr>
<td>Write event</td>
<td>0x0005</td>
<td>Unsigned1</td>
<td>R</td>
<td>R</td>
<td>0x00: no write event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: write event</td>
</tr>
<tr>
<td>Read event</td>
<td>0x0005</td>
<td>Unsigned1</td>
<td>R</td>
<td>R</td>
<td>0x00: no read event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: read event</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>Type</td>
<td>Access</td>
<td>Access</td>
<td>Value</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>----------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>reserved</td>
<td>0x0005</td>
<td>Unsigned1</td>
<td>R</td>
<td>R</td>
<td>0x00</td>
</tr>
<tr>
<td>Mailbox state</td>
<td>0x0006</td>
<td>Unsigned1</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>
|                  |         |          |        |        | 0x00: mailbox empty  
|                  |         |          |        |        | 0x01: mailbox full  |
| Buffered stat    | 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 FDI | 0x0007  | Unsigned1| R      | RW     | 0x00: channel disabled  
|                  |         |          |        |        | 0x01: channel enabled  |
| RepeatAck        | 0x0007  | Unsigned1| R      | RW     |         |
|                  |         |          |        |        | shall follow repeat after data recovery  |
| reserved         | 0x0007  | Unsigned4| R      | RW     | 0x00    |
7.6 Section [FMMU]

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

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

Meaning:

<table>
<thead>
<tr>
<th>LogAddr (Offs)</th>
<th>Len</th>
<th>LogStartBit</th>
<th>LogEndBit</th>
<th>PhysStartBit</th>
<th>PhysAddr</th>
<th>RdWrEnable</th>
<th>ChEnable</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00 00 01</td>
<td>00 00 00 0D 08</td>
<td>00 01</td>
<td>00 01</td>
<td>00 00 00 00</td>
<td>FMMU0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00 00 00 06</td>
<td>00 00 07 11</td>
<td>00 01</td>
<td>00 01</td>
<td>00 00 00 00</td>
<td>FMMU1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>relative address (offset)</th>
<th>Data type</th>
<th>Access type</th>
<th>Access type PDI</th>
<th>Value/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical start address</td>
<td>0x0000</td>
<td>DWORD</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>0x0004</td>
<td>WORD</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Logical start bit</td>
<td>0x0006</td>
<td>Unsigned3</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>0x0006</td>
<td>Unsigned5</td>
<td>RW</td>
<td>R 0x00</td>
<td></td>
</tr>
<tr>
<td>Logical end bit</td>
<td>0x0007</td>
<td>Unsigned3</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>0x0007</td>
<td>Unsigned5</td>
<td>RW</td>
<td>R 0x00</td>
<td></td>
</tr>
<tr>
<td>Physical start address</td>
<td>0x0008</td>
<td>WORD</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Physical start bit</td>
<td>0x000A</td>
<td>Unsigned3</td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>0x000A</td>
<td>Unsigned5</td>
<td>RW</td>
<td>R 0x00</td>
<td></td>
</tr>
<tr>
<td>Read enable</td>
<td>0x000B</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R</td>
<td>0x00: entity will be ignored for read service 0x01: entity will be used for read service</td>
</tr>
<tr>
<td>Write enable</td>
<td>0x000B</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R</td>
<td>0x00: entity will be ignored for write service 0x01: entity will be used for write service</td>
</tr>
<tr>
<td>reserved</td>
<td>0x000B</td>
<td>Unsigned8</td>
<td>RW</td>
<td>R 0x00</td>
<td></td>
</tr>
<tr>
<td>Enable</td>
<td>0x000C</td>
<td>Unsigned1</td>
<td>RW</td>
<td>R 0x00: entity not active 0x01: entity active</td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>0x000C</td>
<td>Unsigned15</td>
<td>RW</td>
<td>R 0x0000</td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>0x000E</td>
<td>WORD</td>
<td>RW</td>
<td>R 0x0000</td>
<td></td>
</tr>
</tbody>
</table>
7.7 Section [SDO]

This section contains the binary SDO data of the device:

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

Meaning:

<table>
<thead>
<tr>
<th>NumServ</th>
<th>Cmd</th>
<th>Index</th>
<th>SubIndex</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 20</td>
<td>2F 12 1C 00 00 00 00 00</td>
<td></td>
<td></td>
<td>&lt;- COE Cmd0</td>
</tr>
<tr>
<td>00 20</td>
<td>2F 13 1C 00 00 00 00 00</td>
<td></td>
<td></td>
<td>&lt;- COE Cmd1</td>
</tr>
<tr>
<td>00 20</td>
<td>2B 13 1C 01 00 1A 00 00</td>
<td></td>
<td></td>
<td>&lt;- COE Cmd2</td>
</tr>
<tr>
<td>00 20</td>
<td>2B 13 1C 02 01 1A 00 00</td>
<td></td>
<td></td>
<td>&lt;- COE Cmd3</td>
</tr>
<tr>
<td>00 20</td>
<td>2F 13 1C 00 02 00 00 00</td>
<td></td>
<td></td>
<td>&lt;- COE Cmd4</td>
</tr>
</tbody>
</table>
SDO Header Word and Command Byte

<table>
<thead>
<tr>
<th>Frame part</th>
<th>Data Field</th>
<th>Data Type</th>
<th>Value/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANopen Header</td>
<td>Number</td>
<td>Unsigned0</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Unsigned3</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>Service</td>
<td>Unsigned4</td>
<td>0x02: SDO Request</td>
</tr>
<tr>
<td>SDO</td>
<td>Size Indicator</td>
<td>Unsigned1</td>
<td>0x00: size of Data (1..4) unspecified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x01: size of Data in Data Set Size specified</td>
</tr>
<tr>
<td></td>
<td>Transfer Type</td>
<td>Unsigned1</td>
<td>0x01: Expedited transfer</td>
</tr>
<tr>
<td></td>
<td>Data Set Size</td>
<td>Unsigned2</td>
<td>0x00: 4 Octet Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x01: 3 Octet Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x02: 2 Octet Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x03: 1 Octet Data</td>
</tr>
<tr>
<td></td>
<td>Complete Access</td>
<td>Unsigned1</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>Command</td>
<td>Unsigned3</td>
<td>0x01: Initiate Download Request</td>
</tr>
</tbody>
</table>

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)
7.7.1 PDO Mapping

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

PDO mapping by DS402

The diagram illustrates the PDO mapping process for PDOs (Process Data Objects) assigned to different SYNCMANs (Synchronous MANagement) and the related mailbox input/output operations. The mapping is performed through SDO transactions, which are represented as arrows connecting the application layer (CoE Rx PDO, CoE Tx PDO) with the data link layer (Process Data).

- **Rx PDO Assign**: 
  - **SYNCMAN 2**: 0x1C12
  - **SYNCMAN 3**: 0x1C13

- **Tx PDO Assign**: 
  - **SYNCMAN 2**: 0x1600
  - **SYNCMAN 3**: 0x1A00

- **Rx PDO Mapping**: 0x600
- **Tx PDO Mapping**: 0x6064
- **Control Word**: 0x6040
- **Status Word**: 0x6041
- **Velocity Control**: 0x606C
- **Position Control**: 0x607A
- **Torque Control**: 0x6077
<table>
<thead>
<tr>
<th>RX-PDO Mapping</th>
<th>TX-PDO Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td><strong>Variables</strong></td>
</tr>
<tr>
<td>0x1600</td>
<td>Controlword</td>
</tr>
<tr>
<td></td>
<td>Mode of Operation (Control)</td>
</tr>
<tr>
<td></td>
<td>Halt Option Code</td>
</tr>
<tr>
<td></td>
<td>Target position</td>
</tr>
<tr>
<td></td>
<td>Target Velocity</td>
</tr>
<tr>
<td></td>
<td>Target Velocity</td>
</tr>
<tr>
<td></td>
<td>Velocity Offset</td>
</tr>
<tr>
<td></td>
<td>Velocity Offset</td>
</tr>
<tr>
<td></td>
<td>Target Torque</td>
</tr>
<tr>
<td></td>
<td>Torque Offset</td>
</tr>
<tr>
<td></td>
<td>Touch Probe (Control)</td>
</tr>
<tr>
<td></td>
<td>Touch Probe 1 Positive Edge</td>
</tr>
<tr>
<td></td>
<td>Touch Probe 1 Negative Edge</td>
</tr>
</tbody>
</table>
7.8 Section [OUTPUT] / [INPUT]

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

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

Meaning (see also ECATCOREDEF.H):

<table>
<thead>
<tr>
<th>Item Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(01 : DATA_ITEM_STATUS)</td>
<td>(01 : DATA_TYPE_U8)</td>
</tr>
<tr>
<td>(02 : DATA_ITEM_VALUE)</td>
<td>(02 : DATA_TYPE_U16)</td>
</tr>
<tr>
<td>(03 : DATA_ITEM_SCALE)</td>
<td>(03 : DATA_TYPE_U32)</td>
</tr>
<tr>
<td>(04 : DATA_ITEM_DIAG)</td>
<td>(04 : DATA_TYPE_U64)</td>
</tr>
<tr>
<td>(05 : DATA_ITEM_NAME)</td>
<td>(05 : DATA_TYPE_I8)</td>
</tr>
<tr>
<td></td>
<td>(06 : DATA_TYPE_I16)</td>
</tr>
<tr>
<td></td>
<td>(07 : DATA_TYPE_I32)</td>
</tr>
<tr>
<td></td>
<td>(08 : DATA_TYPE_I64)</td>
</tr>
<tr>
<td></td>
<td>(09 : DATA_TYPE_F32)</td>
</tr>
<tr>
<td></td>
<td>(0A : DATA_TYPE_F64)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item Type</th>
<th>Data Type</th>
<th>Data Len</th>
<th>FMMU Index</th>
</tr>
</thead>
</table>
| 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
8 EtherCAT Verifier (ECATVERIFY)

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

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

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:

```c
//Enable selected station
for (ULONG i=0; i<_StationNum; i++)
    if (i == m_StationIndex)    __pUserList[i].bDisable = FALSE;
else   __pUserList[i].bDisable = TRUE;
```
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 informational). 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 enabled. Each configuration dialog contains additionally information about the corresponding EtherCAT telegram, which will be sent or received.
This configuration task uses the Library function(s), defined in ECATCOREDEF.H and SHAECATCORE.H:

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

ULONG Result = EcatChangeAllStates(AlState);
```
8.2.1 Configure Station Address

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

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

```
ULONG Result = EcatInitStationAddresses(EcatParams.PhysAddr)
```
8.2.2 Configure FMMU Management

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

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

```c
ULONG Result = EcatInitFmmus(EcatParams.LogicalAddr);
```
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. This configuration task uses the Library function(s), defined in ECATCOREDEF.H and SHAECATCORE.H:

\[
ULONG \text{ Result } = \text{EcatInitSyncManagers}();
\]

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

\[
ULONG \text{ Result } = \text{EcatChangeAllStates(AL\_STATE\_PRE\_OP)};
\]
8.2.4 Configure PDO(s)

The PDOs (Process Data Objects) are typically sent by COE (Can Over Ethernet) with use of mailbox communication. The COE Mailbox communication uses SDOs (Service data Objects) to provide the PDO information to the device. Thus the native format describes SDOs instead of PDO data. The PDO (SDO) configuration dialog allows parsing XML information, EEPROM (SII) information and the Native format for configuration and provides information to all items (also described in the EtherCAT specification).
Each SDO information can be selected by “Left Mouse Double Click” on the corresponding line. When the SDO is selected it can be sent to the device. When all SDO information is sent, configuration task is fulfilled. This configuration task uses the Library function(s), defined in ECATCOREDEF.H, ECATSDODEF.H and SHAECATCORE.H:

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

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

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

```c
__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 EtherCAT 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.PAR (to be placed in \WINDOWS\SYSTEM32). Therefore 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. It's also possible to convert the whole XML file to the native format. Devices which are already present in ECATDEVICE.PAR will be updated.

![Converter XML → File](image)

- **Name**: EL3102
- **Vendor ID (hex)**: 00000002
- **Product Code (hex)**: 0x1e3052
- **Rev. Number (hex)**: 00000000

- **Force PDO Assignment**: 
- **Force PDO Configuration**: 

[Convert Device] [Convert File] [Cancel] [OK]
8.6 PDO Configurator

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

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

![DC Configuration](image)

**Result in ECATDEVICE.PAR:**

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

The master library provides several error handling and tracing mechanisms.

9.1 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.

9.1.1 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.

```c
__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;
}
```
9.1.2 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.

```c
__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.

9.2 Debug LOG File

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

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

9.3 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.
10 Related Dokuments

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