TM

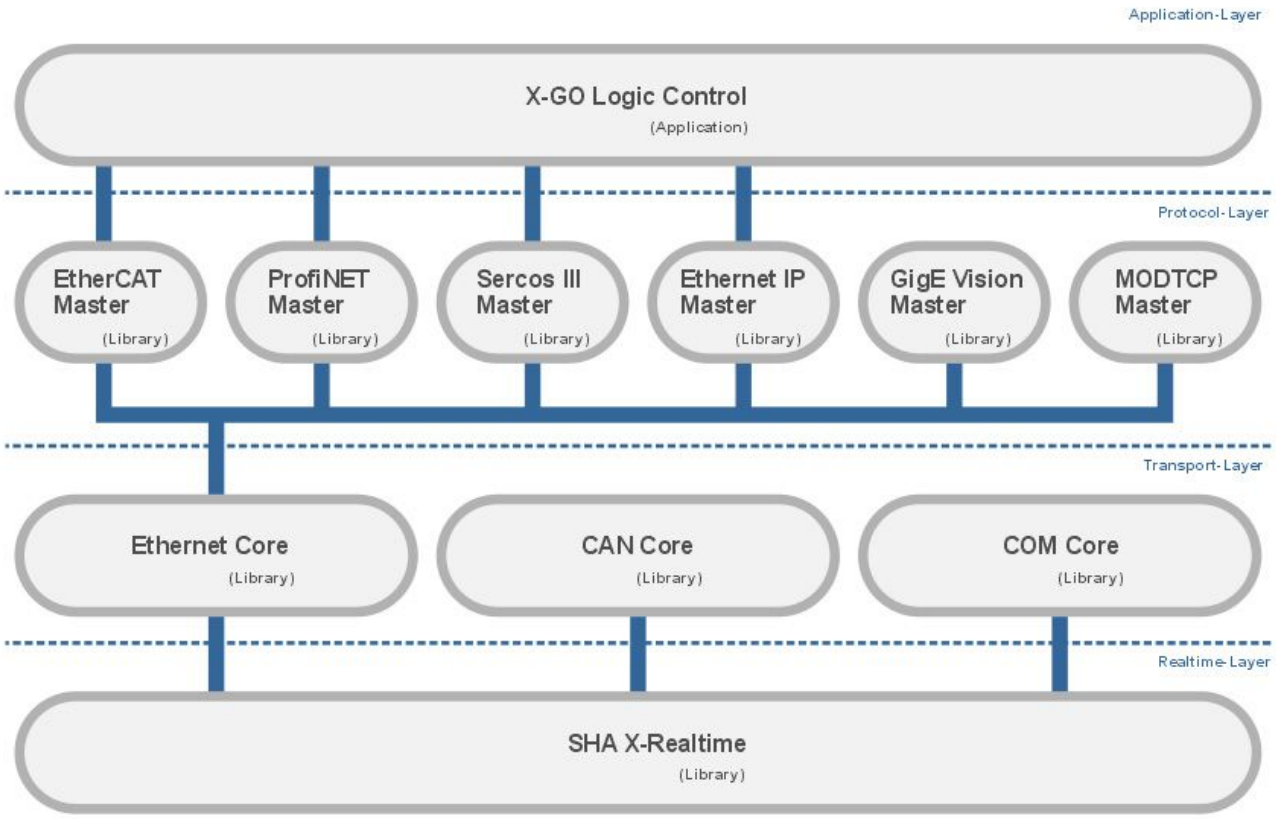
Ethernet/IP Realtime Master Library Documentation (64 Bit)

Date: July,9.2015

Ethernet/IP Realtime Master Library Documentation

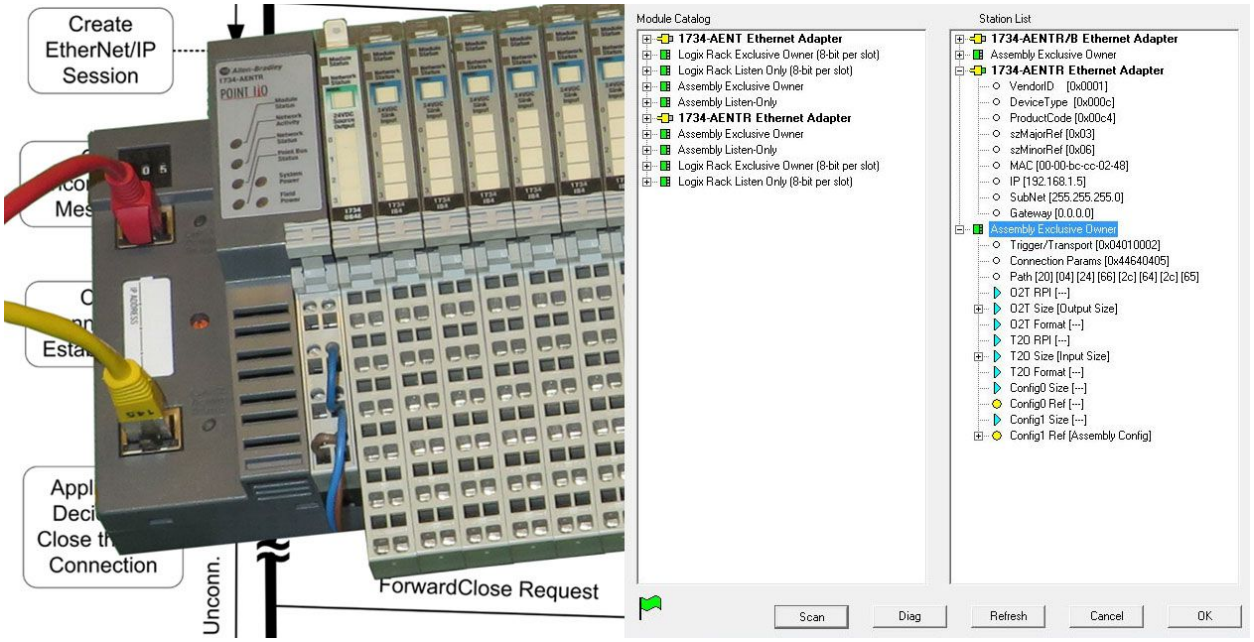
1 Introduction

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



Based on the increasing requirements of networking within the industrial area, the ethernet communication plays a more and more important role for the control of devices. Different ethernet standards are available for the bus related communication, like SERCOS, Powerlink, EtherCAT, Ethernet/IP and Ethernet/IP. The communication principle is based characteristically on a deterministic cyclic data-exchange and requires therefore for devices deterministic Soft- and hardware control.

The SYBERA Ethernet/IP Master library allows the RT control of Ethernet/IP IO devices. The Open Master Library is based on the Ethernet Realtime Core. SYBERA provides therefore a library called SHAEIPCORE for mastering the Ethernet/IP states and sending and receiving of Ethernet/IP frames in realtime. The Open Ethernet/IP Realtime library allows handling of Ethernet/IP data in realtime, without the need of an complex Ethernet/IP management, nevertheless with the ability of interfacing all Ethernet/IP interface levels.



The direct control of the fieldbus devices with a PC as controller hardware and the operating system windows became relevant with the introduction of so-called realtime extensions. An example therefor is the realtime extension of SYBERA that enables ethernet update cycles upto 50 µsec. The Ethernet based bus-communication differs not only through a different protocol-specification, but rather also through the bus-topology. While EtherCAT realizes a ring-topology, Ethernet/IP realizes a star-topology. The Ethernet/IP realtime communication prefaces a defined Master/Slave-relation. With the new realtime Ethernet/IP master the necessity of a separate controller-hardware falls. The Ethernet/IP control can be realized simply by the PC and standard Ethernet adapters.

1.1 Product Features

• Ethernet/IP Configurator
• Multi Connection Management
• Support of Realtime Class 0 and 1
• Sampling upto 100 usec
• Ethernet/IP Service Interface
• High-/Low-Level interface
• Error Management
• Diagnostics
• Sequence Log

1.2 Supported Platforms

- Visual C++ (from Version 8)
- CVI LabWindows

1.3 Supported OS

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

1.4 Reference Devices

- Allen Bradley AENT/R 1734
- Allen Bradley OB4E
- Allen Bradley IB4E
- Bürkert Valveisland 8640

2 Ethernet/IP 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 rights

2. Check the installed Ethernet adapter has given a correct IP address

Installation

3. Install SHA realtime system (separate software package)

4. Install ETH transport library (separate software package)

5. Run the program SYSETUP64 of the Ethernet/IP 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 SYLICENCECHECK64.EXE

Operation

7. Run EIPCONFIG64.EXE

8. Build the program with the library interface

9. Run the program

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

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

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

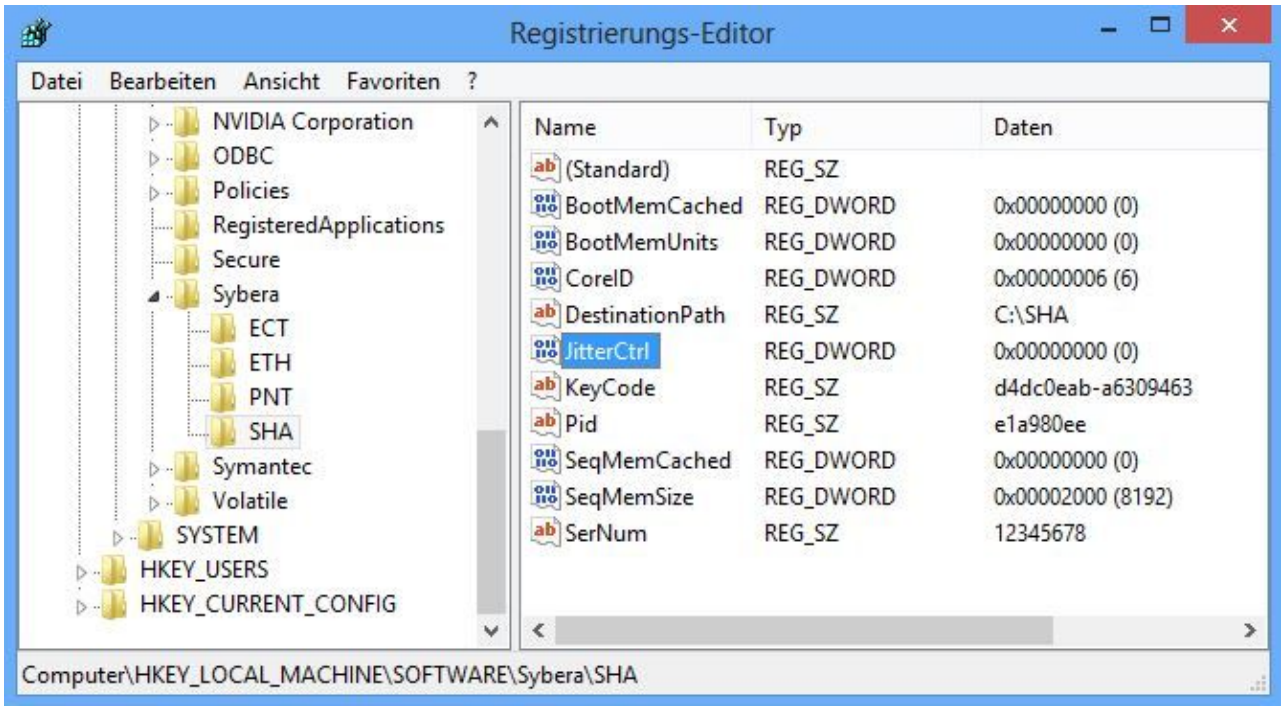
Enhanced SpeedStep — SpeedStep also modulates the CPU clock speed and voltage according to load, but it is invoked via another mechanism. The operating system must be aware of SpeedStep, as must the system BIOS, and then the OS can request frequency changes via ACPI. SpeedStep is more granular than C1E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

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

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

2.1 Jitter Control

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

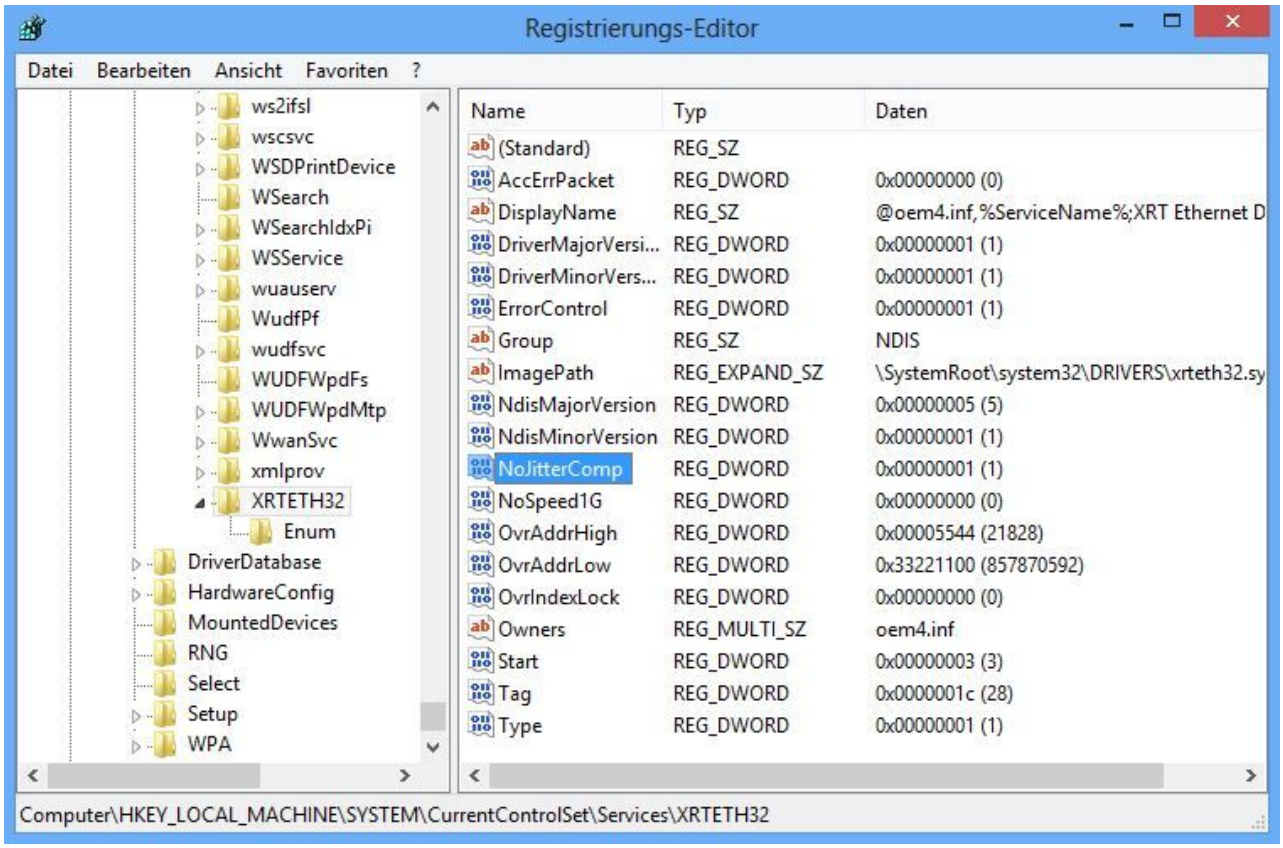


Following values are valid:

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

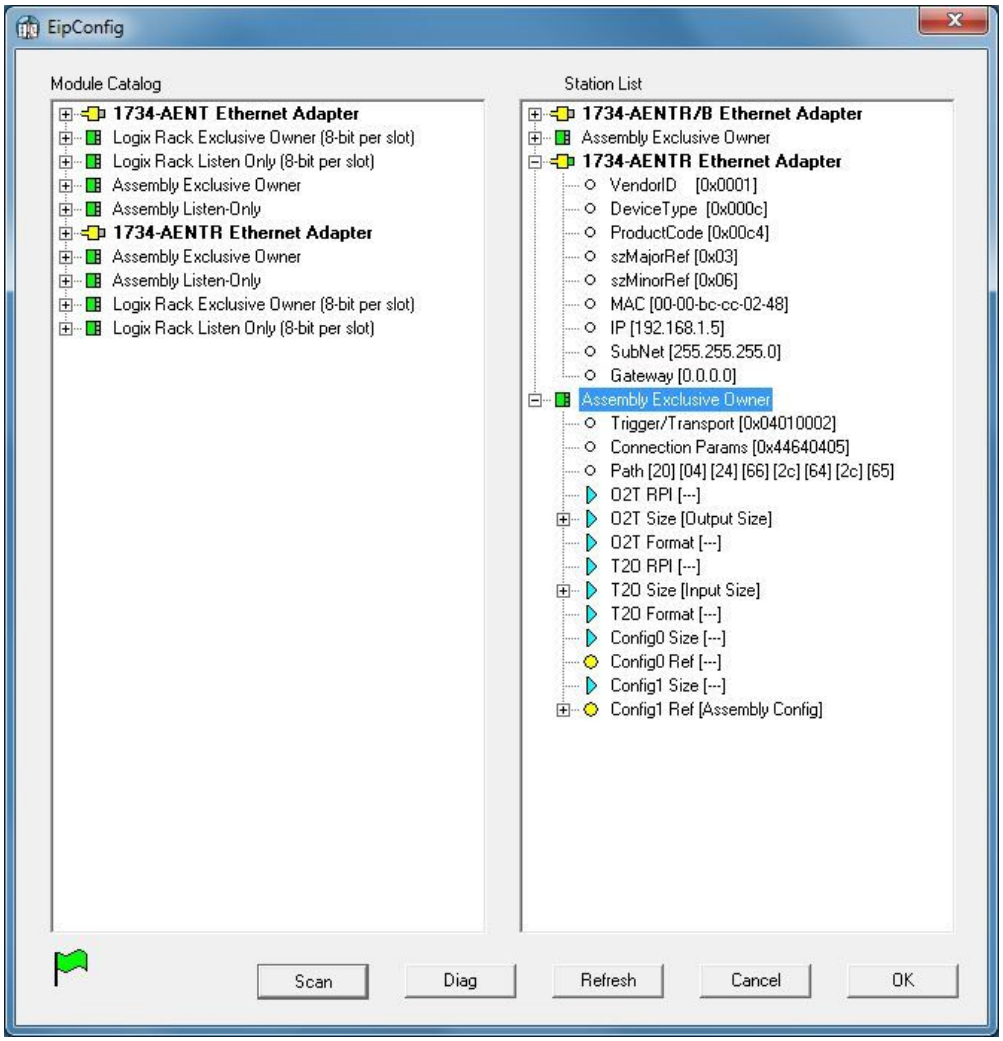
2.2 Dynamic Jitter Compensation

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



3 Creating a Stationlist

A Ethernet/IP fieldbus system consists of several station devices (typically buscoupler devices). A station consists at least of one module (SLOT) and a module consists at least of one submodule (SUBSLOT). For proper operation the Ethernet/IP devices needs first to be configured (by Station Name and IP) and a native STATIONLIST for operating the Ethernet/IP realtime library has to be created. Therefore SYBERA provides a program called EIPCONFIG.EXE.



Note: Make shure a valid IP address is provided for the network connection.

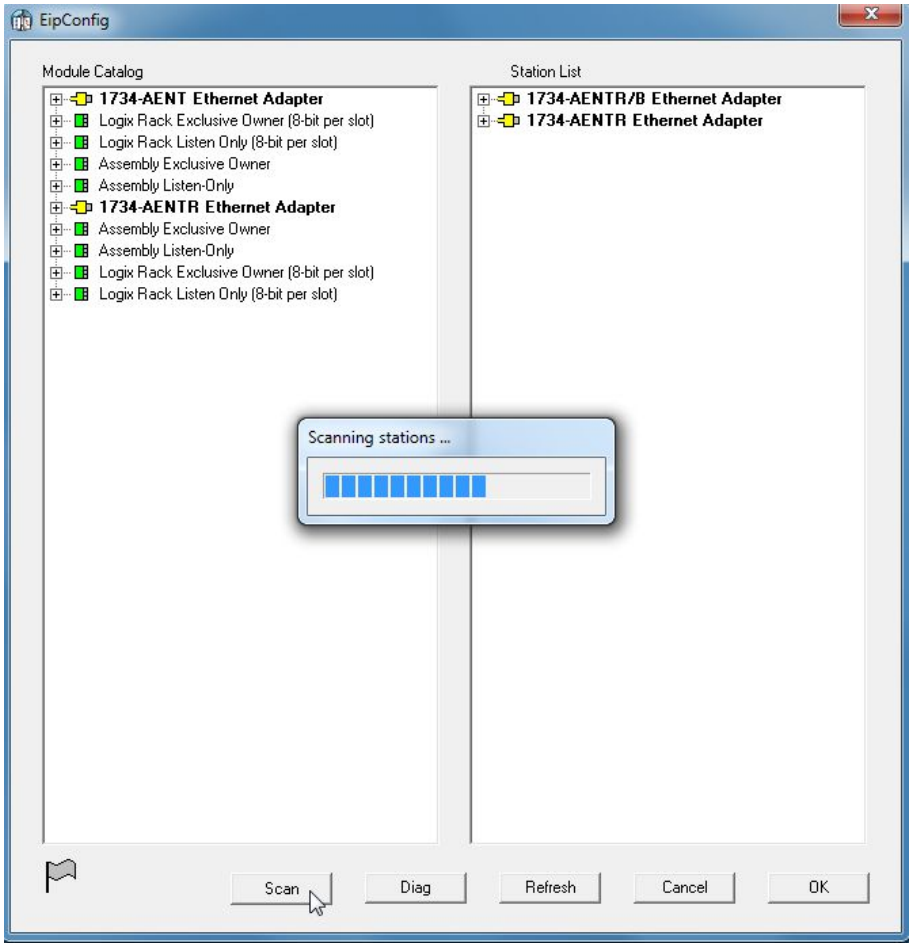
Note: If the application fails to run, check if the lastest Microsoft XML Parser has been installed. If not, install in the directory \APP\MSXML\MSXML4

EIPCONFIG allows creating a native stationlist by selecting connection modules from a module catalog (leftside view). The catalogue get its entries by provided EDS files, which must be present in the same directory as EIPCONFIG. A module is inserted to the station list configuration (rightside view) by a DRAG and DROP operation (just drag a module from the catalog to the station list configuration). There are two types of modules:

- Device Module
- Connection Module

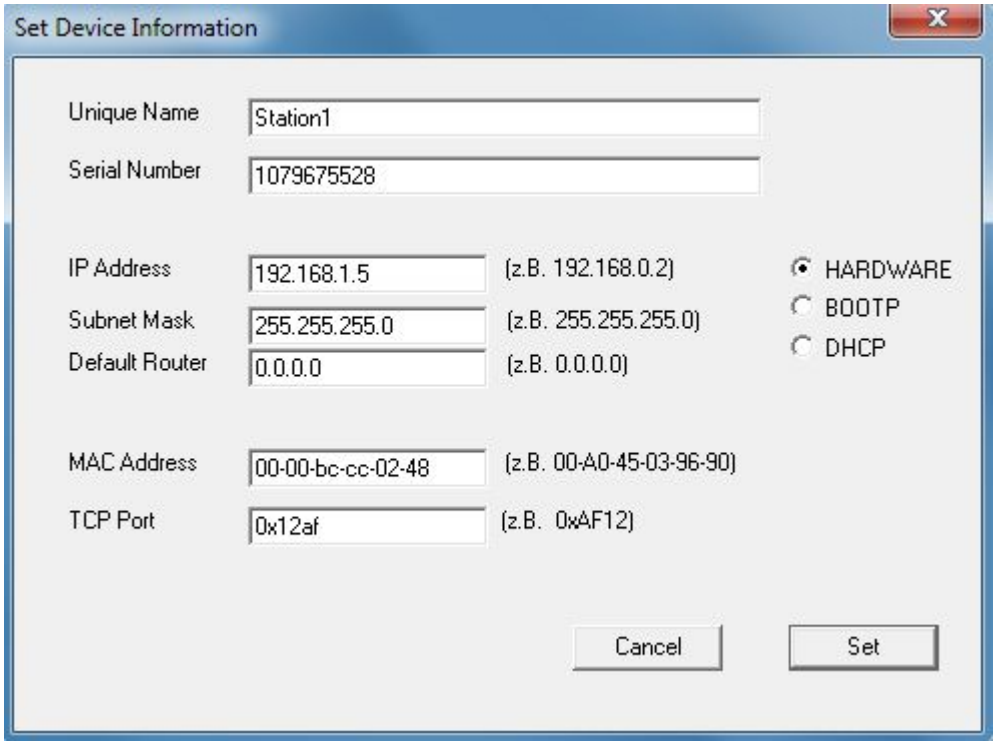
3.1 Device Module

The device module keeps all information gathered from scanning the fieldbus, such as station name, IP parameters, MAC address and other information. Therefore first task is to collect information about the Ethernet/IP configuration by scanning the bus.



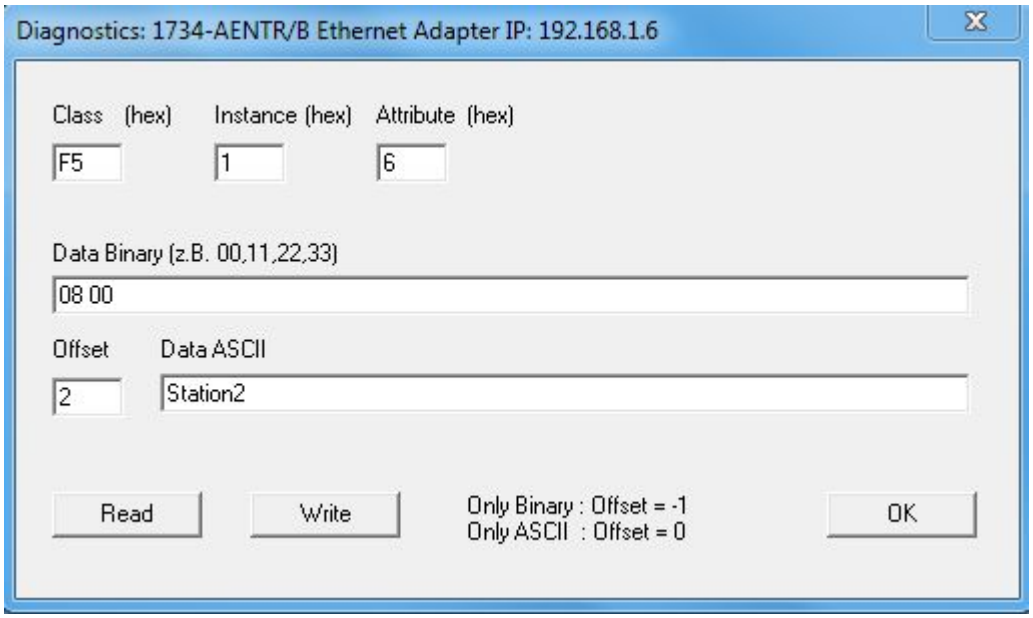
3.2 Station Settings

The scan gets information about manufacturer name and MAC address. Now individual assignment must set (e.g. IP address, station name). On a right button click at the device module the property dialog appears.



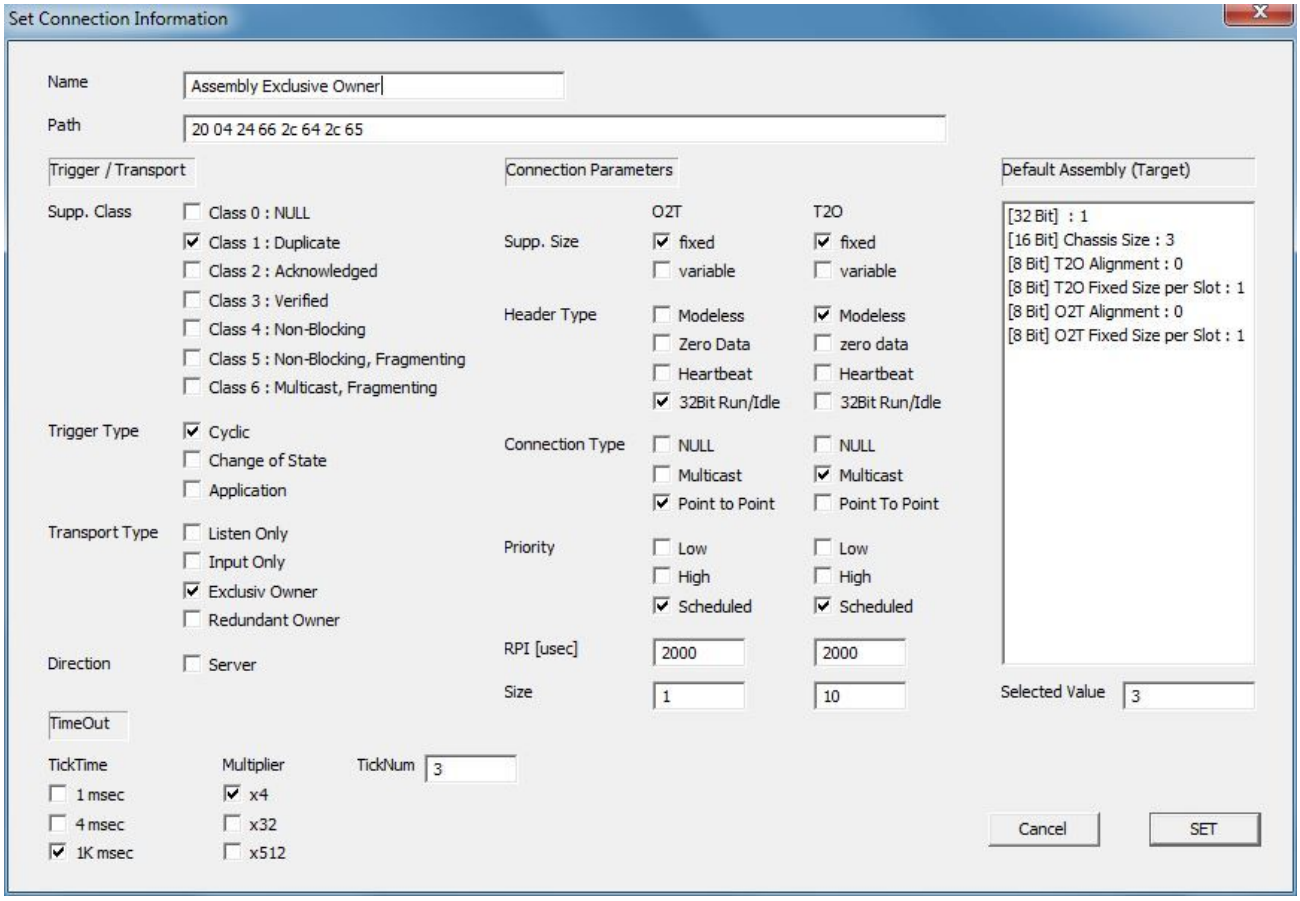
3.3 Station Diagnostics

The station diagnostics allows reading and writing of single attributes. Therefore select a device module and press button DIAG.



3.4 Connection Module

Each station typically consists of multiple connection modules. Connection modules have to be inserted from the catalog by DRAG and DROP operations. A station configuration must contain all functional modules in the correct order (in the order these modules are physically connected). When inserting a module from the catalog, after dropping, it appears at the end of the configuration list and must be pushed to the correct location. On a right button click at the connection module the property dialog appears.



The resulting stationlist is stored in a text file which must be provided to the Ethernet/IP Master Library.

Sample:

```
> Device

[NAME]
1734-AENTR Ethernet Adapter
[SERIAL_NUM]
4e 61 bc 00
[TCP_PORT]
12 af
[MAC_ADDR]
00 11 22 33 44 55
[IP_PARAMS]
c0 a8 01 05 ff ff ff 00 00 00 00 00
[VENDOR_ID]
01 00
[DEVICE_TYPE]
0c 00
[PRODUCT_CODE]
c4 00
[REVISION]
03 02

>> Connection

[NAME]
Assembly Exclusive Owner
[TICK_TIME]
0a
[TICK_NUM]
02
[MULTIPLIER]
00
[TRANSPORT]
01
[O2T_PARAMS]
07 48
[O2T_FORMAT]
04
[O2T_RPI]
10 27 00 00
[T2O_PARAMS]
0c 28
[T2O_FORMAT]
00
[T2O_RPI]
10 27 00 00
[CON_PATH]
20 04 24 66 2c 64 2c 65
```

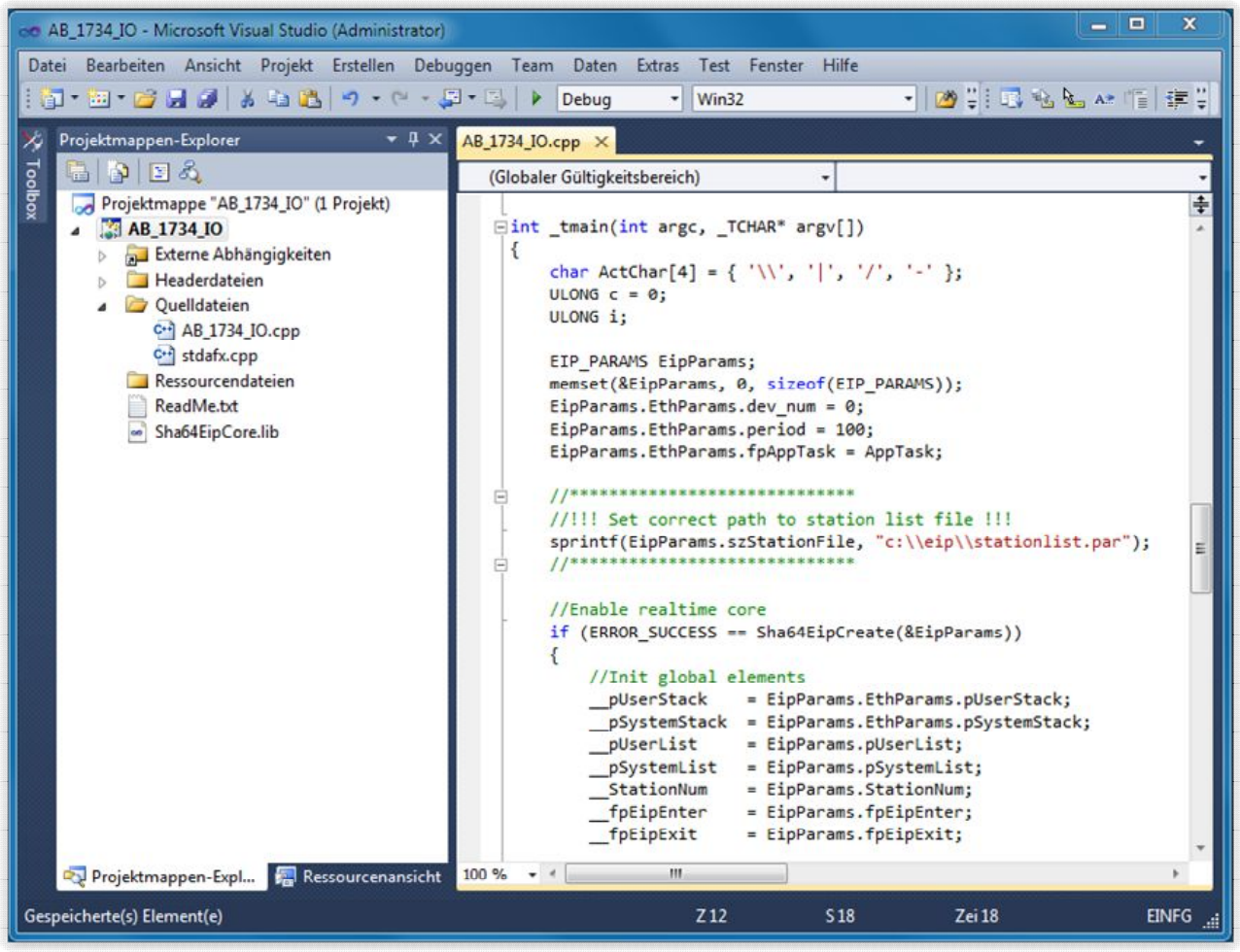
```
[ASSEMBLY_PATH]
80 05 01 00 00 00 03 00 00 01 00 01

> End



//Set stationlist file path
sprintf(EipParams.szStationFile, "c:\\eip\\stationlist.par");
```

4 Ethernet/IP Realtime Master Library

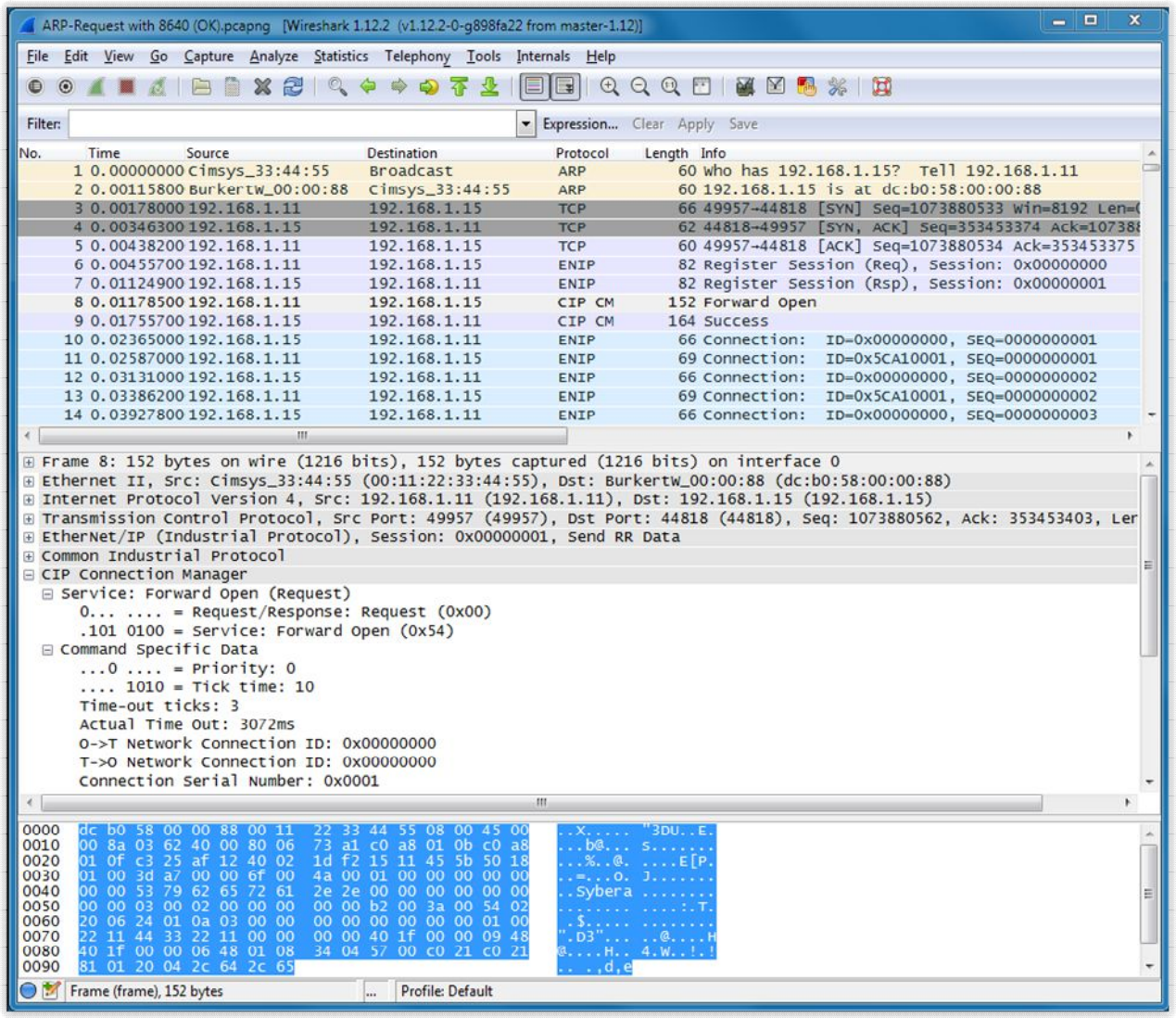
The interface functions of the Ethernet/IP Realtime Master Library are exported by a dynamic link library. Following header files and libraries are required:

SHA*64*EIPCORE.DLL	Ethernet/IP Master DLL (VISUAL C++)
SHA*64*EIPCORE.LIB	Ethernet/IP Master LIB (VISUAL C++)
SHAEIPCOREOML.DLL	Ethernet/IP Master DLL (BORLAND C++ / Delphi)
SHAEIPCOREOML.LIB	Ethernet/IP Master LIB (BORLAND C++ / Delphi)
SHA*64*EIPCORE.H	Exported Function Prototypes
EIP*64*COREDEF.H	Ethernet/IP Basic Definitions
EIP*64*MACROS.H	Ethernet/IP Macro Definitions
STATIONLIST.PAR	Native Station List (generated by EIPCONFIG)
EIPDBG.LOG	Sequence Log (generated at runtime)

Sample Project



Sample Startup Protocol:



4.1 Header File EIP*64*COREDEF.H

The header file EIPCOREDEF.H declares all required structures when handling Ethernet/IP interface functions or handling the Core Realtime Stack directly (Realtime Level2).

4.1.1 Structure EIP_PARAMS

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

```
typedef struct _EIP_PARAMS
{
      //Input parameters
      char              szStationFile[MAX_PATH_SIZE]; //Station list file name
      UCHAR             HostIP[IP_ADDRESS_LEN];       //HOST IP address

      //Output parameters
      ULONG             ErrCnts;                 //Error Counters
      FP_EIP_ENTER      fpEipEnter;              //Function Pointer to EipEnter()
      FP_EIP_EXIT       fpEipExit;               //Function Pointer to EipExit()
      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
      ULONG             StationNum;              //Station Number
      PSTATION_INFO     pSystemList;             //PSTATION_INFO structure for
                                                 //realtime application task
      PSTATION_INFO     pUserList;               //PSTATION_INFO structure for
                                                 //windows application task
} EIP_PARAMS, *PEIP_PARAMS;
```

Note:

The structure ETH_PARAMS is part of the Ethernet Core Library and described in the the documentation of this core libraray. 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 Ethernet realtime core.

4.1.2 Structure STATION_INFO

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

```
typedef struct _STATION_INFO
{
      char        UniqueName[MAX_PATH_SIZE];    //Unique device name
      char        ProductName[MAX_PATH_SIZE];   //Description of device
      USHORT      VendorID;                     //Device manufacturers Vendor ID
      USHORT      DeviceType;                   //Device Type of product
      USHORT      ProductCode;                  //Product Code assigned with
                                                //respect to device type
      TYPE16      Revision;                     //Device revision
                                                //(Major / Minor Revision)
      ULONG       SerialNum;                    //Serial number of device
      ULONG       SessionHandle;                //Session Handle
      TYPE64      SenderContext;                //Sender Context
      BOOLEAN     bCipEncap;                    //Support of CIP packet
                                                //encapsulation
      BOOLEAN     bCipClassUdp;                 //Support of CIP Class 0/1
                                                //connections
      UCHAR       MacAddr[ETH_ADDRESS_LEN];     //MAC Address
      IP_PARAMS   IpParams;                     //IP parameters
      UINT_PTR    TcpSock;                      //TCP socket
      TYPE16      TcpPort;                      //TCP port
      USHORT      Status;                       //Current status of device
      USHORT      ConNum;                       //Number of connections
      STATION_CON ConList[MAX_CON_NUM];         //Connection list
      UCHAR       O2tData[MAX_FRAME_DATA];      //O2T Data
      UCHAR       T2oData[MAX_FRAME_DATA];      //T2O Data
      ULONG       State;                        //Station State
      ULONG       Error;                        //Station Error

} STATION_INFO, *PSTATION_INFO;
```

Note:

The most elements of the structure STATION_INFO will be automatically filled with the provided Stationlist information. The elements InputFrameData and OutputFrameData keep the payload data of the station.

4.1.3 Structure STATION_CON

This structure keeps all information of each Ethernet/IP connection and may be required for further interface functions.

```
typedef struct _STATION_CON
{
      ULONG             ConTime;                //Connection Time
      USHORT            ConSerNum;              //Connection Serial Number
      UCHAR             Multiplier;             //Used to calculate request
                                                //timeout information
      UCHAR             TickNum;                //Used to calculate request
                                                //timeout information
      CON_TIMING        Timing;                 //Network Timing Parameters
      CON_TRANSPORT     Transport;              //Network Transport Parameters
      CON_PARAMS        O2tParams;              //O2T Connection Parameters
      UCHAR             O2tFormat;              //O2T Transport Format
      ULONG             O2tRpi;                 //O2T Packet Rate Interval [usec]
      ULONG             O2tSize;                //O2T Payload Data Size
      ULONG             O2tConID;               //O2T connection ID
      CON_PARAMS        T2oParams;              //T2O Connection Parameters
      UCHAR             T2oFormat;              //T2O Transport Format
      ULONG             T2oRpi;                 //T2O Packet Rate Interval [usec]
      ULONG             T2oSize;                //T2O Payload Data Size
      ULONG             T2oConID;               //T2O connection ID
      UCHAR             ConPath[MAX_PATH_SIZE];       //Connection path data
      ULONG             ConPathLen;                   //Connection path length
      UCHAR             AssemblyPath[MAX_PATH_SIZE];  //Assembly path data
      ULONG             AssemblyPathLen;              //Assembly path length

} STATION_CON, *PSTATION_CON;
```

4.2 Header File EIP*64*MACROS.H

This header file defines all macros required for handling the realtime Task

//Macro to get EIP output data pointer

```
__inline PUCHAR __GetO2tPayloadPtr(
                              PSTATION_INFO pStation, //Station information
                              ULONG ConIndex)         //Connection index
```

//Macro to get EIP input data pointer

```
__inline PUCHAR __GetT2oPayloadPtr(
                              PSTATION_INFO pStation, //Station information
                              ULONG ConIndex)         //Connection index
```

Sample:

```
void static AppTask(PVOID)
{
     ULONG ConIndex;
     ULONG State;

     //Check if initialization is done
     if (__bInitDone == FALSE)
          return;

     //************************************
     //Call EIP enter function
     PSTATION_INFO pStation = __fpEipEnter(&ConIndex, &State);
     if (pStation)
     {
     //************************************

          //Check station state
          if (State == EIP_STATE_VALID)
          {
               //Check station unique name
               if (strcmp(pStation->UniqueName, "Station1") == 0)
               {
                   PUCHAR pInData  = __GetT2oPayloadPtr(pStation, ConIndex);
                   PUCHAR pOutData = __GetO2tPayloadPtr(pStation, ConIndex);

                   //Set input data to output data
                   pOutData[0] = pInData[0];
               }

               //Increase update counter
               __UpdateCnt++;
          }
          else
               //Set station state
               pStation->State = EIP_STATE_ERROR;
     }

     //************************************
     //Call EIP exit function
     __fpEipExit();
     //************************************
}
```

4.3 Debug Log File

The Ethernet/IP master library provides a buildin log sytem which produces a debug log file called *EIPDBG.LOG*. This file contains nessecary information of the library sequence.

Sample:

```
EIPCORE64 -> Sha64EipCreate

EIPCORE64 -> CreateStationList
C:\XGO\StationList (AB1734).par

EIPCORE64 -> LoadStationList
C:\XGO\StationList (AB1734).par

EIPCORE64 -> DrvCreate

EIPCORE64 -> Sha64EipGetVersion

EIPCORE64 -> DrvGetVersion

EIPCORE64 -> Eip64EnableStation
Station2

EIPCORE64 -> Eip64EnableStation
Station1

EIPCORE64 -> Eip64DisableStation
Station2

EIPCORE64 -> Eip64DisableStation
Station1

EIPCORE64 -> Sha64EipDestroy

EIPCORE64 -> DrvDestroy

EIPCORE64 -> DestroyStationList
C:\XGO\StationList (AB1734).par
```

5 Ethernet/IP Library Interface

The header file SHAEIPCORE.H defines all required prototypes and parameters of the Ethernet Core Library. The header file is based on the files RAWCOREDEF.H and ETHCOREDEF.H. 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++, Borland C++ Builder and CVI Lab Windows
DP : Borland Delphi

5.1 Basic Functions

5.1.1 Sha*64*EipCreate

This function initializes the Ethernet/IP module states. On success the returning value is ERROR_SUCCESS, otherwise the returning value corresponds to that with GetLastError().

VC
```
ULONG Sha64EipCreate(PEIP_PARAMS);
```

5.1.2 Sha*64*EipDestroy

This function closes the Ethernet/IP communication.

VC
```
ULONG Sha64EipDestroy(PEIP_PARAMS);
```

5.1.3 Sha*64*EipGetVersion

This function retrieves the version information of the Ethernet/IP Master Library, the Ethernet Core Library, the Ethernet Core Driver, the SHA Dll, the SHA Library and the SHA Driver.

VC
```
ULONG Sha64EipGetVersion(PEIP_PARAMS);
```

Sample:

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "c:\ETH\eth32coredef.h"
#include "c:\ETH\sha32ethcore.h"
#include "c:\EIP\Sha32EipCore.h"
#include "c:\EIP\Eip32CoreDef.h"
#include "c:\EIP\Eip32Macros.h"

#pragma warning(disable:4996)


PETH_STACK      __pUserStack = NULL;
PETH_STACK      __pSystemStack = NULL;
PSTATION_INFO   __pUserList = 0;          //PSTATION_INFO structure for
                                          //windows application task
PSTATION_INFO   __pSystemList = 0;        //PSTATION_INFO structure for
                                          //realtime application task
ULONG           __StationNum = 0;
FP_EIP_ENTER    __fpEipEnter = NULL;
FP_EIP_EXIT     __fpEipExit  = NULL;
ULONG           __UpdateCnt = 0;
BOOLEAN         __bInitDone = FALSE;       //Initialization flag

UCHAR           __InData = 0;
UCHAR           __OutData = 0;


void static AppTask(PVOID)
{
     ULONG ConIndex;
     ULONG State;

     //Check if initialization is done
     if (__bInitDone == FALSE)
          return;

     //*************************************
     //Call EIP enter function
     PSTATION_INFO pStation = __fpEipEnter(&ConIndex, &State);
     if (pStation)
     {
     //*************************************

          //Check station state
          if (State == EIP_STATE_VALID)
          {
               //Check station unique name
               if (strcmp(pStation->UniqueName, "Station1") == 0)
               {
                   PUCHAR pInData  = __GetT2oPayloadPtr(pStation, ConIndex);
                   PUCHAR pOutData = __GetO2tPayloadPtr(pStation, ConIndex);
```

```
                //Set input data to outputdata
                pOutData[0] = __OutData = __InData = pInData[0];
            }

            //Increase update counter
            __UpdateCnt++;
        }
        else
            //Set station state
            pStation->State = EIP_STATE_ERROR;
    }

    //*************************************
    //Call EIP exit function
    __fpEipExit();
    //*************************************
}


//*****************************************
//*** MAIN section ************************
//*****************************************

int _tmain(int argc, _TCHAR* argv[])
{
      char ActChar[4] = { '\\', '|', '/', '-' };
      ULONG c = 0;
      ULONG i;

      EIP_PARAMS EipParams;
      memset(&EipParams, 0, sizeof(EIP_PARAMS));
      EipParams.EthParams.dev_num = 0;
      EipParams.EthParams.period = 100;
      EipParams.EthParams.fpAppTask = AppTask;

      //****************************
      //!!! Set correct path to station list file !!!
      sprintf(EipParams.szStationFile, "c:\\eip\\stationlist.par");
      //****************************

      //Enable realtime core
      if (ERROR_SUCCESS == Sha32EipCreate(&EipParams))
      {
            //Init global elements
            __pUserStack      = EipParams.EthParams.pUserStack;
            __pSystemStack    = EipParams.EthParams.pSystemStack;
            __pUserList       = EipParams.pUserList;
            __pSystemList     = EipParams.pSystemList;
            __StationNum      = EipParams.StationNum;
            __fpEipEnter      = EipParams.fpEipEnter;
            __fpEipExit     = EipParams.fpEipExit;

            //Set initialization flag
            __bInitDone = TRUE;

            //Get version information
            Sha32EipGetVersion(&EipParams);
```

```
            //Display enumerated stations
            printf("Number of Stations: %i\n", EipParams.StationNum);
            for (i=0; i<EipParams.StationNum; i++)
                  printf("%i: %s - %s\n", i,
                        __pUserList[i].UniqueName,
                        __pUserList[i].ProductName);

            //Enable stations
            for (i=0; i<EipParams.StationNum; i++)
                  Eip32EnableStation(&__pUserList[i].IpParams);

            //Wait for key pressed
            printf("\nPress any key ...\n");
            while (!kbhit())
            {
                  //Print input payload information
                  printf("In:[0x%02x] State:[%i] - UpdateCnt:[%i] [%c]\r",
                        __InData,
                        __pUserList[0].State,
                        __UpdateCnt, ActChar[++c%4]);

                  //Do some delay
                  Sleep(100);
            }

            //Disable stations
            for (i=0; i<EipParams.StationNum; i++)
                  Eip32DisableStation(&__pUserList[i].IpParams);

            //Destroy EIP realtime core
            Sha32EipDestroy(&EipParams);
      }
}
```

5.2 CORECMD functions

The low level interface provides all function to handle Ethernet/IP core commands

5.2.1 Eip64ScanToFile

Scan devices from bus to file

VC
```
ULONG Result = Eip64ScanToFile(char* pszFile);
```

5.2.2 Eip64SetStationIpControl

Set IP configuration Control (stored value, BOOTP, DHCP)

VC
```
ULONG Result = Eip64SetStationIpControl(
                                        PIP_PARAMS pParams,
                                        ULONG IpControl);
```

5.2.3 Eip64SetStationIP

Set station IP address

VC
```
ULONG Eip64SetStationIP(
                            PIP_PARAMS pParams,
                            PUCHAR pIpAddr);
```

Note:

```
//Structure for IP parameters (declared in ETHCOREDEF.H)
typedef struct _IP_PARAMS
{
      TYPE32      ip_addr;
      TYPE32      subnet;
      TYPE32      gateway;

} IP_PARAMS, *PIP_PARAMS;
```

5.2.4 Eip64SetStationName

Set station name

VC
```
ULONG Result = Eip64SetStationName(
                                    PIP_PARAMS pParams,
                                    char* pszName);
```

5.2.5 Eip64EnableStation

Enable station

VC
```
ULONG Eip64EnableStation(PIP_PARAMS pParams);
```

5.2.6 Eip64DisableStation

Disable station

VC
```
ULONG Eip64DisableStation(PIP_PARAMS pParams);
```

5.3 ENCAP Functions

The low level interface provides all function to handle Ethernet/IP encapsulation services

5.3.1 Eip64ListIdentify

Scan all station information from bus

```
VC ULONG Result = Eip64ListIdentify(
                                    PSTATION_INFO pStationList,
                                    PULONG pStationNum,
                                    ULONG MaxStationNum)
```

5.3.2 Eip64ListServices

List station capability flags

```
VC ULONG Result = Eip64ListServices(PSTATION_INFO pStation)
```

5.3.3 Eip64RegisterSession

Register session – This service must be first at all other services

```
VC ULONG Result = Eip64RegisterSession(PSTATION_INFO pStation);
```

5.3.4 Eip64UnregisterSession

Unregister Session – This service closes the active session

```
VC ULONG Result = Eip64UnregisterSession(PSTATION_INFO pStation);
```

5.3.5 Eip64SendRRData

Send RR Data – This service encapsulates CIP commands.

```
VC ULONG Result = Eip64SendRRData(
                                    PSTATION_INFO pStation,
                                    PUCHAR pReqPacket,
                                    USHORT ReqPacketLen,
                                    PUCHAR pRespPacket,
                                    PUSHORT pRespPacketLen,
                                    USHORT MaxRespPacketSize,
                                    LONG TimeoutSec)
```

5.4 CIP Functions

The low level interface provides all function to handle Ethernet/IP encapsulation services

5.4.1 Eip64ServiceGetAttributeSingle

This service gets a single device attribute.

```
VC ULONG Result = Eip64ServiceGetAttributeSingle(
                                        PSTATION_INFO pStation,
                                        PUCHAR pCipResp,
                                        UCHAR ClassID,
                                        UCHAR InstanceID,
                                        UCHAR AttribID,
                                        PUCHAR pData,
                                        PUSHORT pDataLen,
                                        USHORT MaxDataSize,
                                        LONG TimeoutSec)
```

5.4.2 Eip64ServiceSetAttributeSingle

This service sets a single device attribute.

```
VC ULONG Result = Eip64ServiceSetAttributeSingle(
                                        PSTATION_INFO pStation,
                                        PUCHAR pCipResp,
                                        UCHAR ClassID,
                                        UCHAR InstanceID,
                                        UCHAR AttribID,
                                        PUCHAR pData,
                                        USHORT DataLen,
                                        LONG TimeoutSec)
```

5.4.3 Eip64ServiceForwardOpen

This service opens the cyclic data exchange

```
VC ULONG Result = Eip64ServiceForwardOpen(
                                              PSTATION_INFO pStation,
                                              PUCHAR pCipResp,
                                              ULONG Index)
```

5.4.4 Eip64ServiceForwardClose

This service closes the cyclic data exchange

```
VC ULONG Result = Eip64ServiceForwardClose(
                                            PSTATION_INFO pStation,
                                            PUCHAR pCipResp,
                                            ULONG Index)
```

Sample:

```
EIP_PARAMS EipParams;
memset(&EipParams, 0, sizeof(EIP_PARAMS));
EipParams.EthParams.dev_num = 0;
EipParams.EthParams.period = 100;
EipParams.EthParams.fpAppTask = NULL;

//****************************
//!!! Set correct path to station list file !!!
sprintf(EipParams.szStationFile, "c:\\eip\\stationlist.par");
//****************************

//Enable realtime core
if (ERROR_SUCCESS == Sha32EipCreate(&EipParams))
{
      PSTATION_INFO pStation = (PSTATION_INFO)&EipParams.pUserList[0];
      char szName[MAX_PATH] = { 0 };
      IP_PARAMS IpParams = { 0 };
      UCHAR MacAddr = { 0 };

      //Register session
      if (ERROR_SUCCESS == Eip32RegisterSession(pStation))
      {
            //Send CIP services
            Eip32ServiceGetAttributeSingle(
                        pStation, NULL,
                        0xF5, 1, 0x06,
                        (PUCHAR)&szName, NULL, MAX_PATH_SIZE, 3);

            Eip32ServiceGetAttributeSingle(
                        pStation, NULL,
                        0xF5, 1, 0x05,
                        (PUCHAR)&IpParams, NULL, sizeof(IP_PARAMS), 3);

            Eip32ServiceGetAttributeSingle(
                        pStation, NULL,
                        0xF6, 1, 0x03,
                        (PUCHAR)MacAddr,   NULL, ETH_ADDRESS_LEN, 3);

            //Unregister session
            Eip32UnregisterSession(pStation);
      }

      //Wait for key pressed
      printf("\nPress any key ...\n");
      while (!kbhit()) { Sleep(100); }

      //Destroy EIP realtime core
      Sha32EipDestroy(&EipParams);
}
```

6 Realtime Operation

After enabling the Ethernet/IP system (*ShaEipCreate*) with a corresponding station list, the realtime tasks become active. The application realtime task is decorated by Realtime Wrapper functions:

```
//Call EIP enter function (Return: pointer to current station)
PSTATION_INFO pStation = __fpEipEnter(
                              &ConIndex,  // OUT: Connection index of station
                              &State);    // OUT: Station state


typedef PSTATION_INFO   (__cdecl *FP_EIP_ENTER)(PULONG, PULONG);
typedef VOID            (__cdecl *FP_EIP_EXIT) (VOID);
```

These wrapper functions are used to manage the realtime Ethernet/IP protocol management, like ethernet frame update, error handling, stack management,… The Ethernet/IP Library Realtime System itself is managed by syncronized states:

```
//Define EIP Wrapper States
enum _EIP_STATES
{
      EIP_STATE_INIT = 0,
      EIP_STATE_ENABLED,
      EIP_STATE_DISABLED,
      EIP_STATE_VALID,
      EIP_STATE_ERROR
};
```

Get valid station state

```
if (pStation->State == EIP_STATE_VALID)
{
      …
};
```

Sample

```
void static AppTask(PVOID)
{
     ULONG ConIndex;
     ULONG State;

     //Check if initialization is done
     if (__bInitDone == FALSE)
          return;

     //************************************
     //Call EIP enter function
     PSTATION_INFO pStation = __fpEipEnter(&ConIndex, &State);
     if (pStation)
     {
     //************************************

          //Check station state
          if (State == EIP_STATE_VALID)
          {
               //Check station unique name
               if (strcmp(pStation->UniqueName, "Station1") == 0)
               {
                    PUCHAR pInData  = __GetT2oPayloadPtr(pStation, ConIndex);
                    PUCHAR pOutData = __GetO2tPayloadPtr(pStation, ConIndex);
               }

               //Increase update counter
               __UpdateCnt++;
          }
          else
               //Set station state
               pStation->State = EIP_STATE_ERROR;
     }

     //************************************
     //Call EIP exit function
     __fpEipExit();
     //************************************
}
```

7 Error Handling

The master library provides an error handling and tracing mechanism.

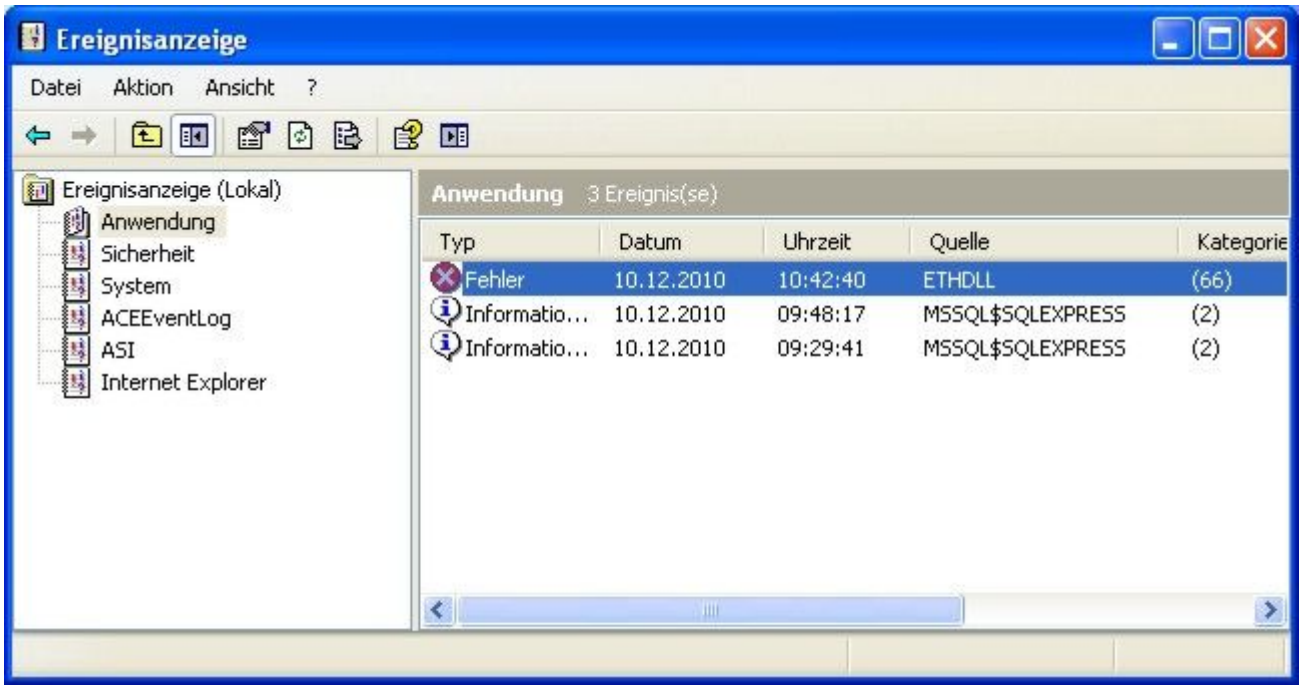
7.1 Debug LOG File

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

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

7.2 Event File

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



8 Related Documents

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