TM

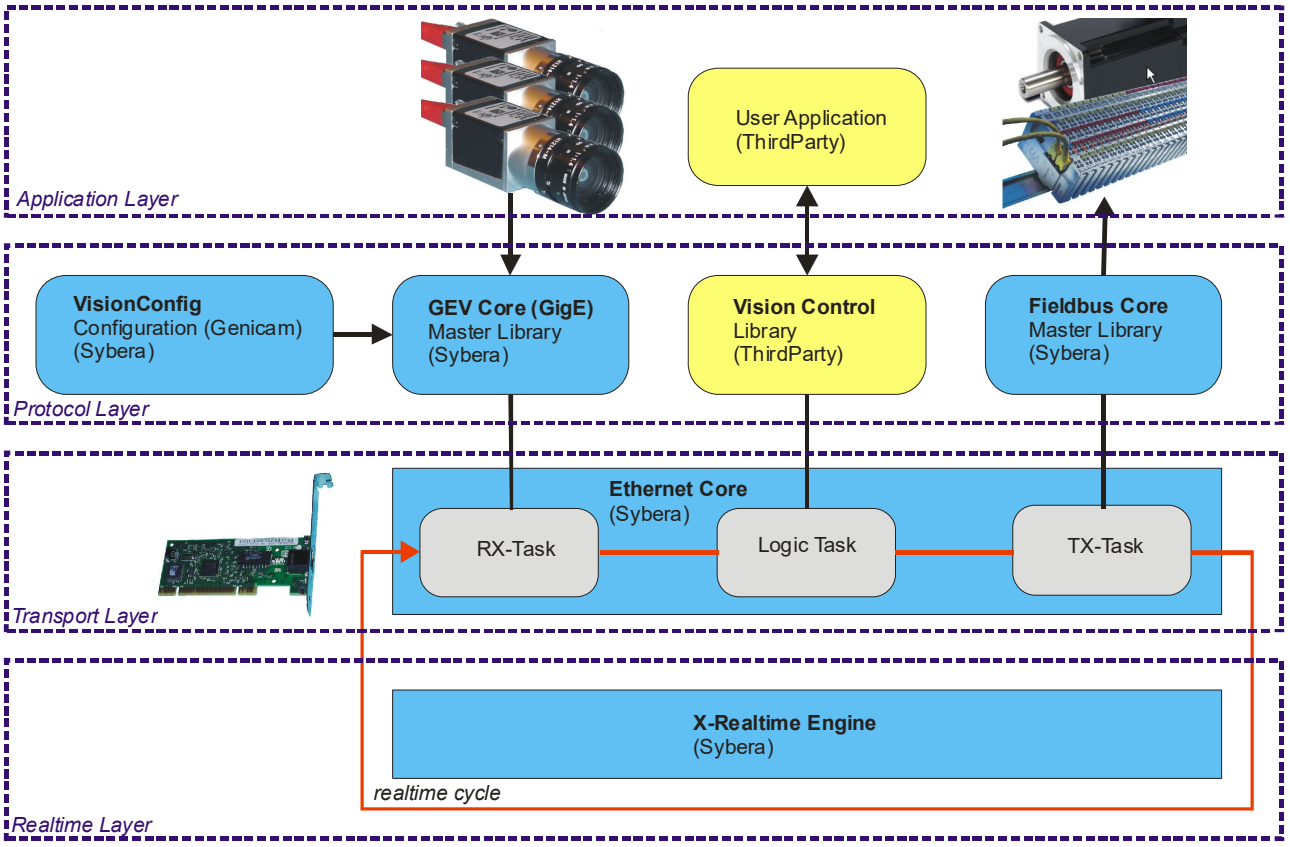
GigE-Vision Realtime Master Library Documentation (64 Bit)

Date: July,9.2015

GigE Vision Realtime Master Library Documentation

1 Introduction

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 and GIGE. The communication principle is based characteristically on a deterministic process data-exchange and requires therefore for devices deterministic Soft- and hardware control.



GigE-Vision is an international interface standard for industrial image processing . It allows easy connection of industrial cameras to existing network systems through the use of Gigabit Ethernet. The GigE-Vision standard consists of four elements : The GigE-Vision Control Protocol (GVCP and GVSP) uses UDP and defines how a device is to be addressed and specifies data channels and mechanisms for the transmission of images and configuration data between camera and PC. The GigE-Vision Stream Protocol (GVSP) specifies the different data types and modes of transmission to transfer images. UDP is used for transmission. But GigE-Vision itself contains the optional possibility of packet resend to correct transmission errors. The GigE-Vision Device Discovery mechanism includes finding cameras on the network. Additionally, the camera manufacturer provides a XML file based on the GeniCam standard that contains a functional description of the camera. This allows access to camera data and images. With an integrated bandwidth control, the GigE-Vision realtime master library allows the complete control cycle of "Vision Sensor - Image Processing - Fieldbus actuators" in a deterministic time grid distributed to different fieldbuses. This is realized by the use of the X-Realtime cluster engine for Windows. With the GigE-Vision SYBERA master, within a single network or across multiple networks data can be exchanged with other fieldbus systems. The camera configuration follows the GenICam standard parameters (e.g. Basler Camera) . The image processing can be performed in realtime with third-party libraries (e.g. OpenCV) . The master control is done directly from the PC and will be implemented with standard Ethernet adapters. By extending the Ethernet transport layer (Ethernet realtime core) more than 70 standard Ethernet adapter are supported for the GigE-Vision connection.

The GigE-Vision master of SYBERA is offered as an open realtime library system for Windows and allows the developer to program a deterministic control for cameras in conjunction with other field bus systems. The developer has the option to program the cameras with simple interface functions, and to create own vision applications graphically. As with the standardized programming interfaces the developer has the possibility to realize the configuration of the logical link and vision cameras by library functions. The developer of vision projects should not have to worry about protocol management, rather take care of the processing of the relevant image data of the connected cameras. In addition, it must also be ensured, that the developer can also take influence to complex operations, such as State change of the cameras or error situations with available library functions. The GigE-Vision protocol management (GVCP and GVSP) , error handling and logistics equipment are implemented in so called Master protocol stacks. The protocol stacks thus forming the connection between the physical transport layer (eg, Ethernet driver) and application software.

The Master meets the requirements of the GigE-Vision specification 2.0 and is designed as other SYBERA master libraries as open programming system. Particularly high value was placed on the critical timing of the GigE-Vision GVSP protocol . This is done through an integrated bandwidth control. The system is based on four realtime tasks for sending and receiving Ethernet frames and functional treatment for cyclic data exchange. The tasks are functionally synchronized by a state machine. The realtime error task detects frame errors and hardware latencies. With help of a frame filter, the acyclic GVCP vision frames are separated in realtime and transfered to a telegram stack. SYBERA implemented in the master libraries the process "Dynamic Jitter Compensation" with active and passive feedback. Although the X-Realtime Engine of SYBERA manages a low maximum jitter of 15 microseconds (depending on the hardware platform), an additive jitter arises in the sampling operation. For unsynchronized communication with Industrial Ethernet components this is negligible in most cases. However, to realize the control and synchronization of drives, the reduction of the additive jitter and dynamic drift in the master libraries was required.

1.1 Product Features

- Windows (32- und 64-Bit)
- Realtime Image Processing
- GigE-Vision Protocol
- GenICam Configuration
- OpenCV compatible
- GVCP acyclic transfer
- GVSP cyclic realtime transfer
- Bandwidth Adjustment
- Sequence Log

1.2 Supported Platforms

- Visual C++ (from Version 8)

1.3 Supported OS

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

1.4 Reference Devices

- Basler acA1300-30gc

2 GigE-Vision 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 GigE-Vision 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 VISIONGEN64.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 carried out 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.

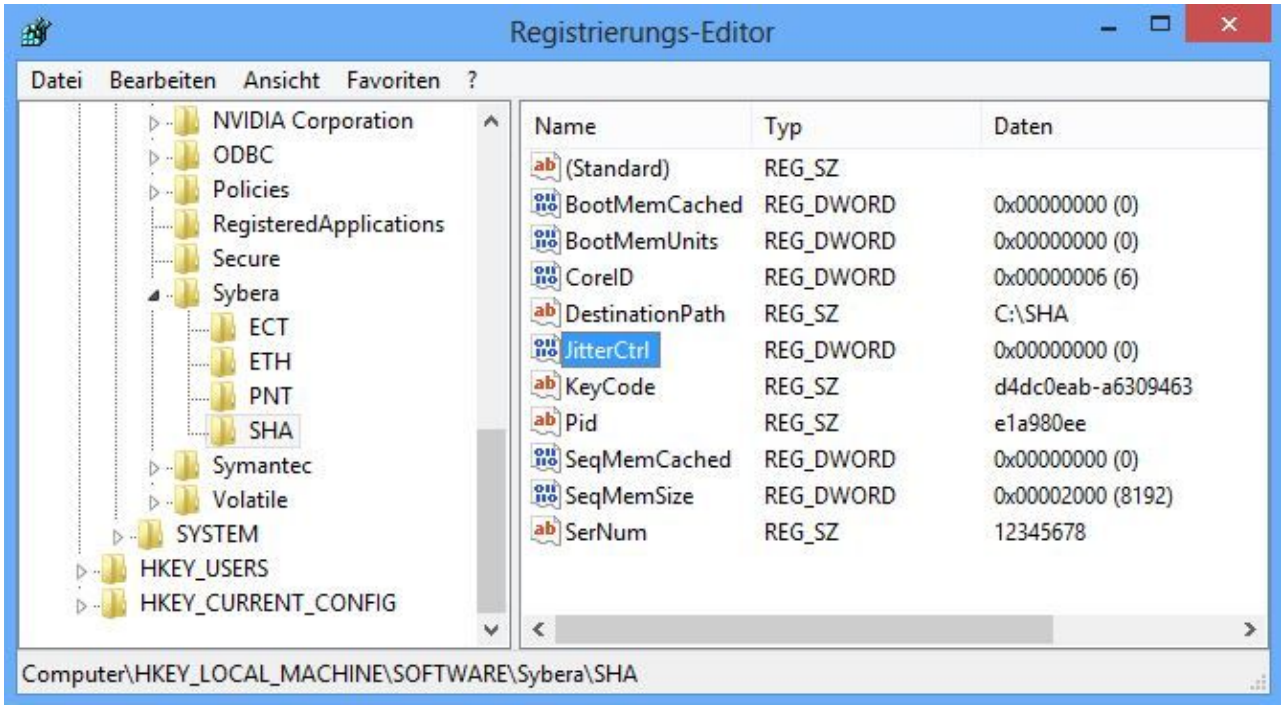
Enhanced SpeedStep — SpeedStep also modulates the CPU clock speed and voltage according to load, but it is invoked via another mechanism. The operating system must be aware of SpeedStep, as must the system BIOS, and then the OS can request frequency changes via ACPI. SpeedStep is more granular than C1E 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.

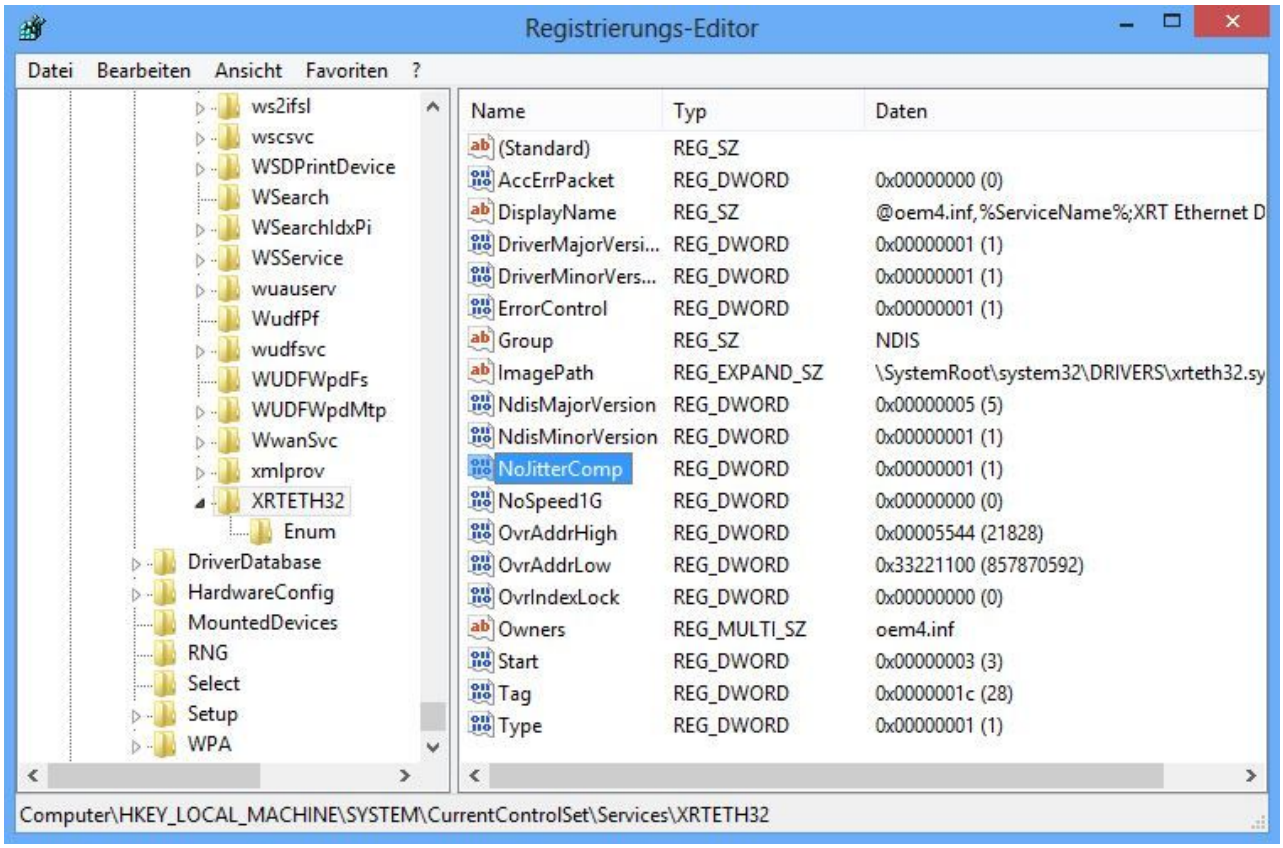


Following values are valid:

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)

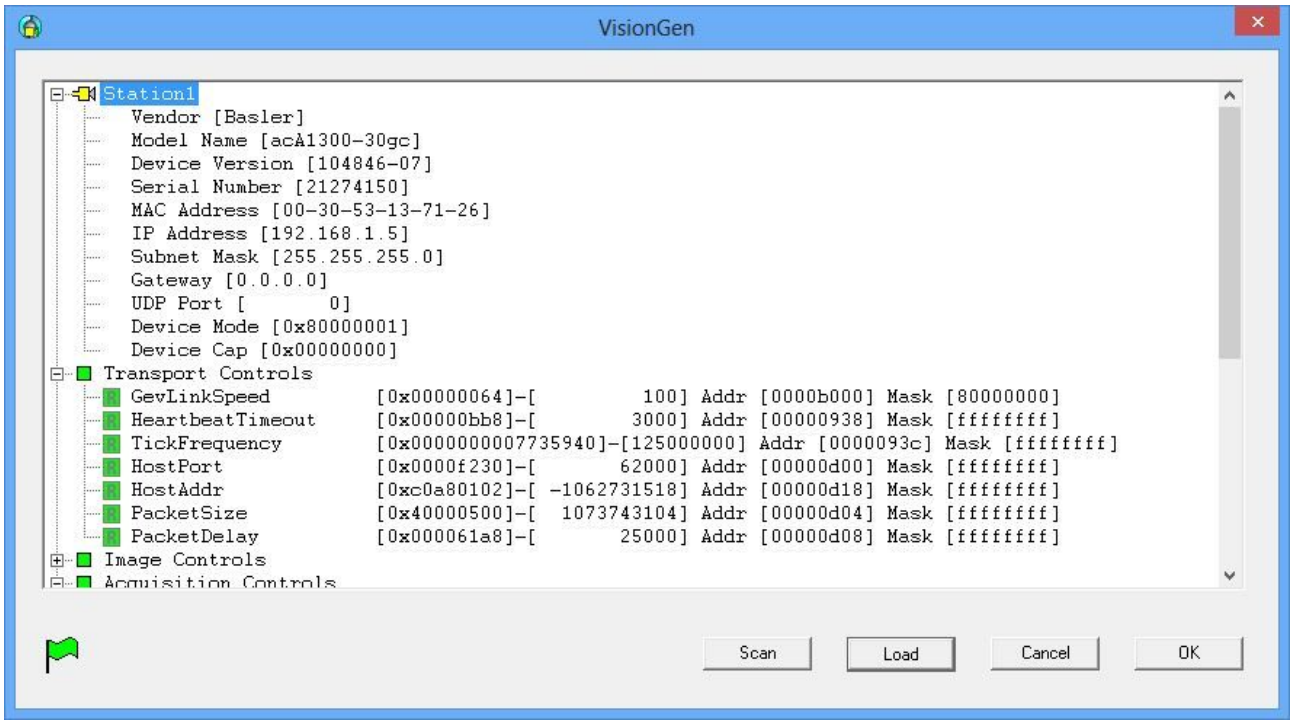
2.2 Dynamic Jitter Compensation

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 Camera Configuration

A GigE-Vision fieldbus system typically consists of multiple (different) camera devices. The configuration of a GigE vision camera is based on the GeniCam vision standard. A camera configuration consists at least of one accesspoint and several Genicam register groups. For proper operation of the SYBERA vision realtime core the GigE-Vision devices need first to be configured by creating a native configuration file. Therefore SYBERA provides a program called VISIONGEN.EXE.



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

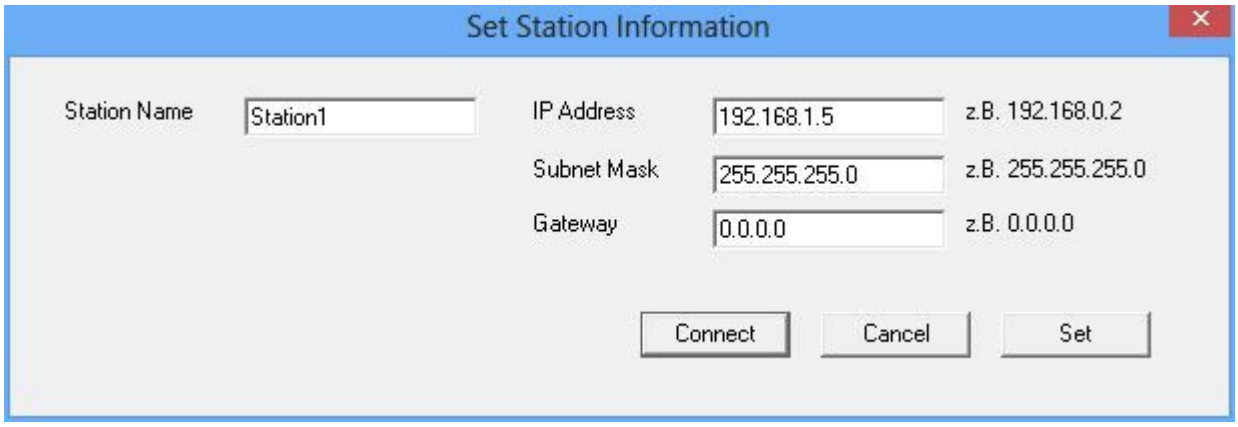
Note: If the application fails to run, check if the lastest Microsoft XML Parser has been installed. If not, install in the directory \MSXML\MSXML6

VISIONGEN allows creating a native configuration file, with all required information for controlling thecamera. This configuration is based on the GeniCam Standard, by selecting camera information from a provided XML camera file which must be present in the same directory as VISIONGEN. The creation of a configuration file can be done by scanning the bus and / or modifying the configuration manually. There are two types of information:

- GigE Accesspoint
- GeniCam Register Group

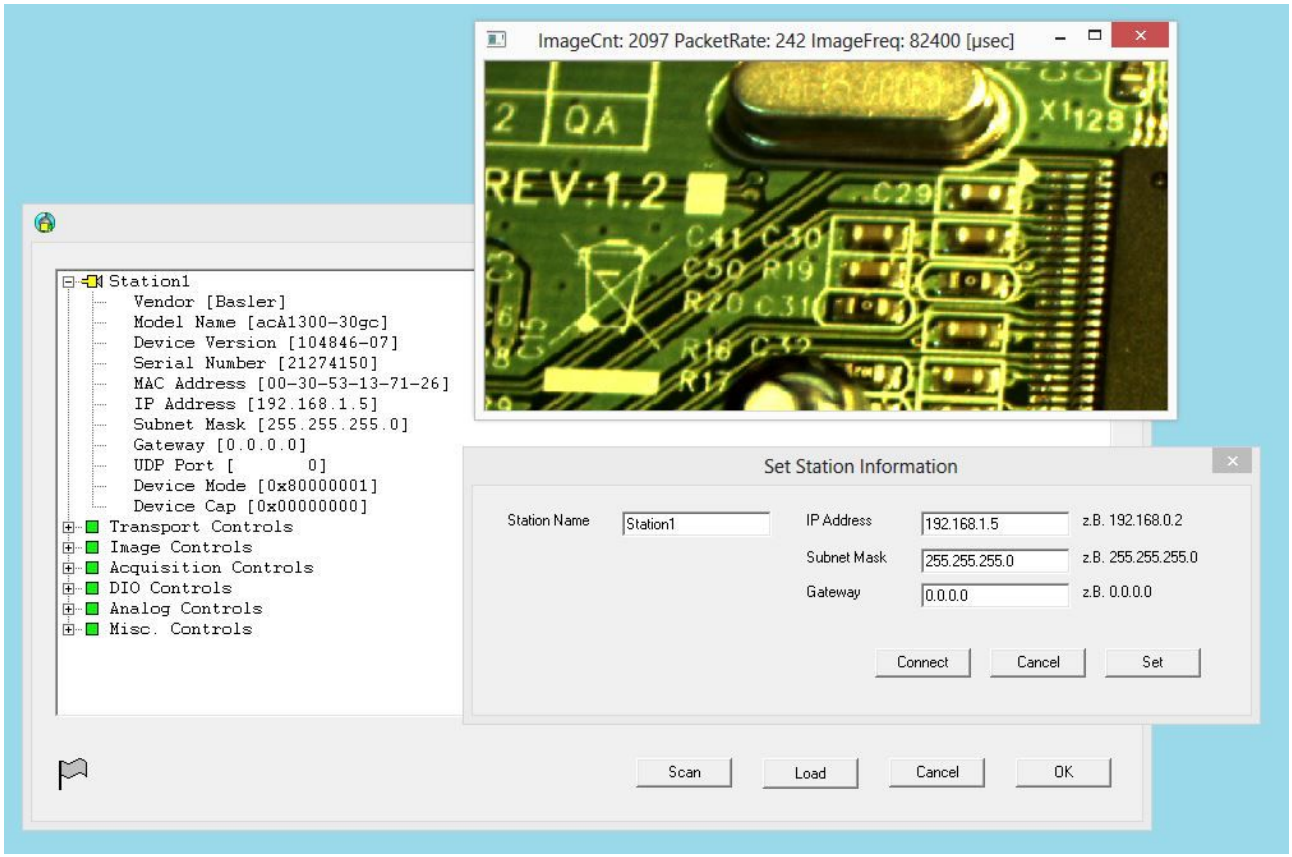
3.1 GigE Accesspoint

The accesspoint keeps all information required for connecting to the fieldbus, as station name, IP parameters, MAC address, identify parameters. Therefore first task is to collect information about the GigE-Vision configuration by scanning the bus. 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 to the selected GigE accesspoint the dialog appears:



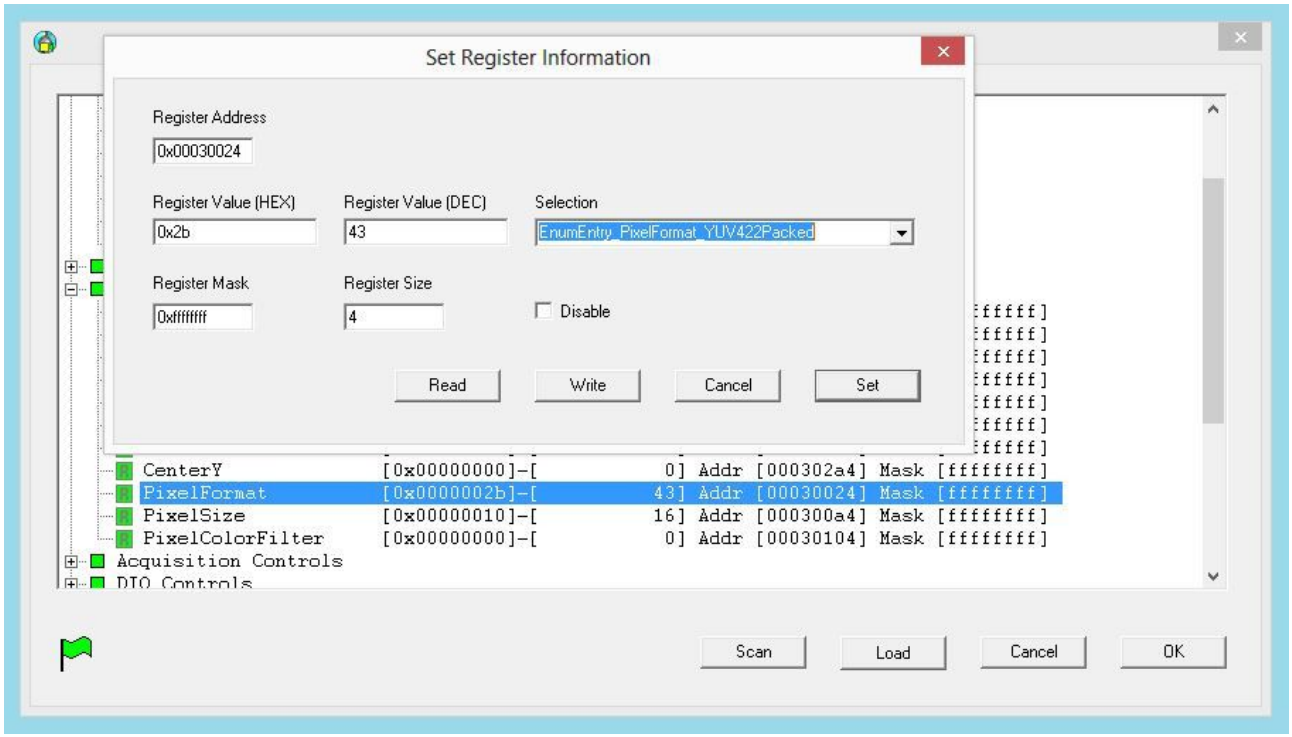
3.1.1 Connecting a Camera

After scanning the bus, or after a configuration has been loaded, the selected camera may be connected



3.2 GeniCam Register Groups

GenICam is the standard for generic camera interfaces. Cameras, which are used in industrial image processing, are configerable more or less in the same way, by reading and writing of parameters to specified control register. However, protocols, formats, and addresses of these registers usually vary from manufacturer to manufacturer. The GeniCam registers are read or written via the ascyclic GVCP protocol standard. On a right button click to the selected GeniCam register the dialog appears:



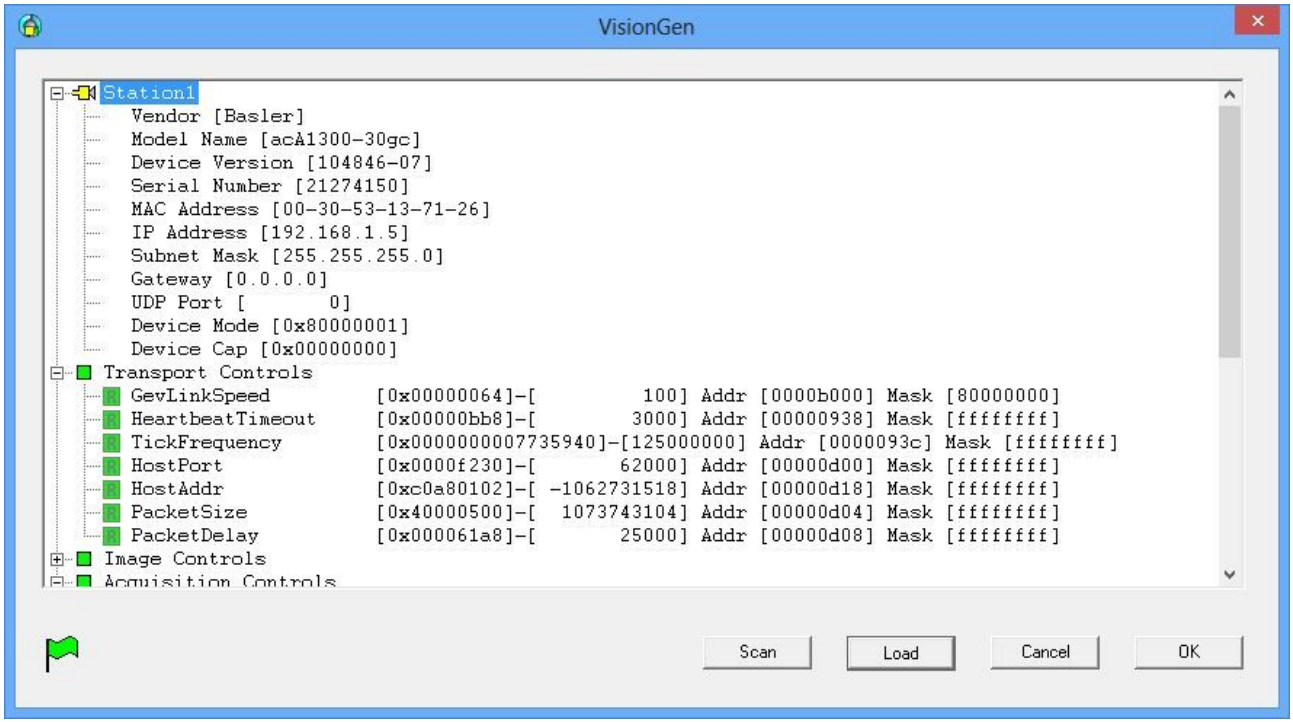
Note: A GeniCam register may also be disabled, so it will be ignored while initializing the camera. To disable a GeniCam register, either setting “Disable” or setting the register size to zero.

Note: Some GeniCam registers allow a control selection with a corresponding value.

Note: The GeniCam registers may be read or written while a camera is currently operating (camera is connected).

At least, typically following GeniCAM registers need to be configured to operate a camera:

AcquisitionMode:	0x00040004
Heartbeat Timeout:	0x0938
Stream Channel Packet Size:	0x0D04
Stream Channel Destination Address:	0x0D18
Stream Channel Port:	0x0D00
Pixel Format:	0x00030024
Image Width:	0x00030204
Image Height:	0x00030224
Offset X:	0x00030244
Offset Y:	0x00030264
Acquisition Start:	0x00040024
PacketSize	0x40000500
GainAuto	0x00000002 (Addr: 0x00020004)



The configuration file is stored in a text format which must be provided to the GigE-Vision Master Library.

Sample:

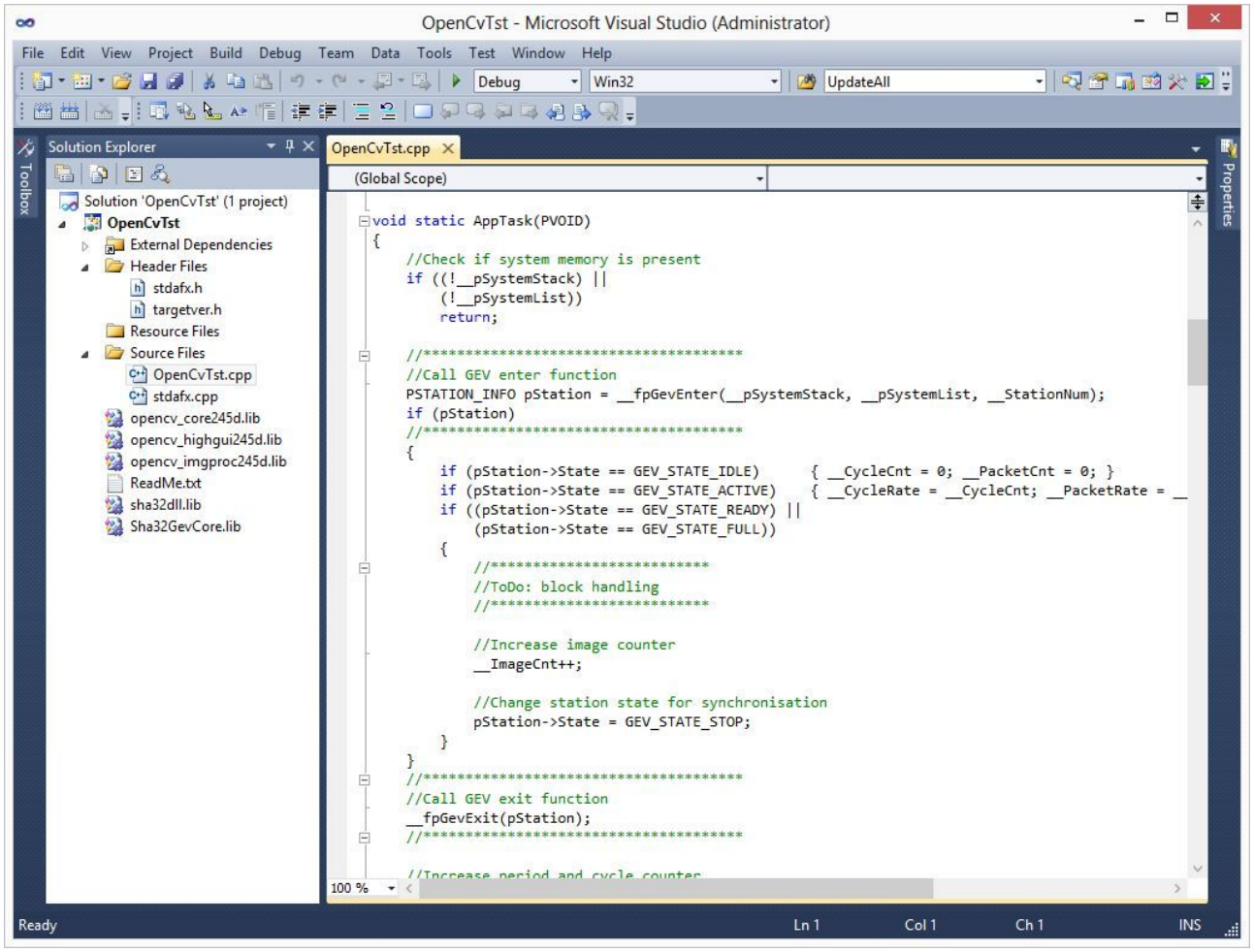
```
> Station

[NAME]
Station1
[MFG_NAME]
Basler
[MODEL_NAME]
acA1300-30gc
[DEV_VERSION]
104846-07
[SER_NUM]
21274150
[MAC_ADDR]
00 30 53 13 71 26
[UDP_PORT]
00 00
[IP_PARAMS]
c0 a8 01 05 ff ff ff 00 00 00 00 00
[DEV_MODE]
01 00 00 80
[DEV_CAP]
00 00 00 00

[LINK_SPEED]
00 b0 00 00 04 00 00 00 00 00 00 80 00 00 00 00 64 00 00 00 00 00 00 00
[HEARTBEAT_TIMEOUT]
38 09 00 00 04 00 00 00 ff ff ff ff 00 00 00 00 b8 0b 00 00 00 00 00 00
[TICK_FREQ]
3c 09 00 00 08 00 00 00 ff ff ff ff 00 00 00 00 40 59 73 07 00 00 00 00
[HOST_PORT]
00 0d 00 00 04 00 00 00 ff ff ff ff 00 00 00 00 30 f2 00 00 00 00 00 00
[HOST_ADDR]
18 0d 00 00 04 00 00 00 ff ff ff ff 00 00 00 00 02 01 a8 c0 00 00 00 00
[PKT_SIZE]
04 0d 00 00 04 00 00 00 ff ff ff ff 00 00 00 00 00 1f 00 40 00 00 00 00
[PKT_DELAY]
08 0d 00 00 04 00 00 00 ff ff ff ff 00 00 00 00 a8 61 00 00 00 00 00 00

…


//Set configuration file path
sprintf(GevParams.szStationFile, "c:\\gev\\stationlist.par");
```

4 GigE-Vision Realtime Master Library

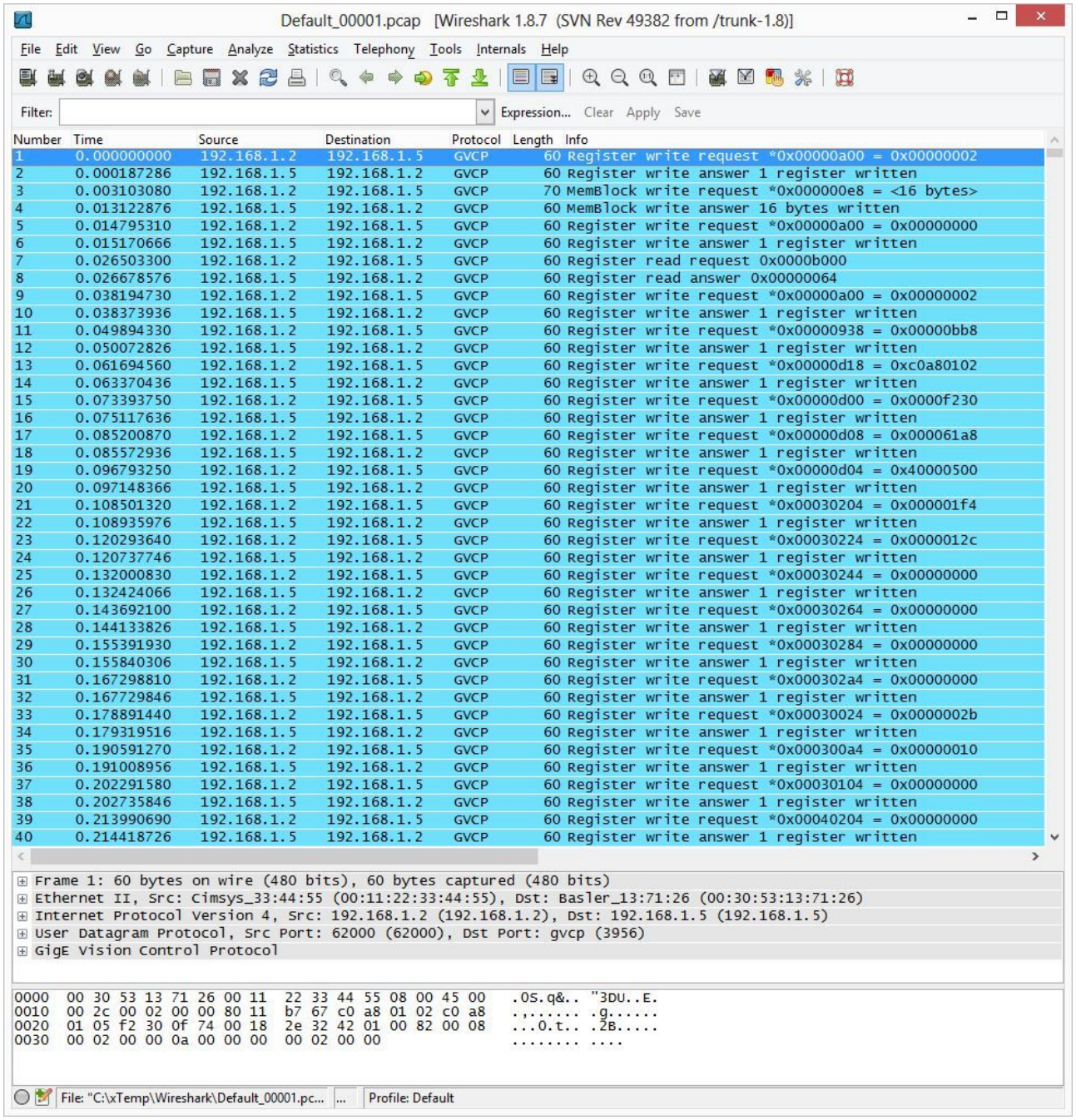
The interface functions of the GigE-Vision Realtime Master Library are exported by a dynamic link library. Following header files and libraries are required:

SHA64GEVCORE.DLL	GigE-Vision Master DLL (VISUAL C++)
SHA64GEVCORE.LIB	GigE-Vision Master LIB (VISUAL C++)
SHA64GEVCORE.H	Exported Function Prototypes
GEV64COREDEF.H	GigE-Vision Basic Definitions
GEV64MACROS.H	GigE-Vision Macro Definitions
GENICAMDEF.H	GeniCam Definitions
GVSPDEF.H	GVSP Definitions
STATIONLIST.PAR	Native Station List (generated by VISIONGEN)
GEVDBG.LOG	Sequence Log (generated at runtime)

Sample Project



Sample Wireshark Protocol:



4.1 Header File GEV64COREDEF.H

The header file GEVCOREDEF.H declares all required structures when handling GigE-Vision interface functions or handling the Core Realtime Stack directly (Realtime Level2).

4.1.1 Structure GEV_PARAMS

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

```
typedef struct _GEV_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_GEV_ENTER      fpGevEnter; //Function Pointer to GevEnter()
      FP_GEV_EXIT fpGevExit;        //Function Pointer to GevExit()
      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
} GEV_PARAMS, *PGEV_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 GigE-Vision modul and may be required for further interface functions.

```
typedef struct _STATION_INFO
{
      STATION_HDR       Hdr;                    //Station Header
      ULONG             State;                  //Station State
      ULONG             Error;                  //Station Error
      ULONG             Index;                  //Station Index

      //Image buffer list
      IMG_BLOCK_INFO    BlockList[MAX_IMG_BLOCKS];    //Image Block List
      ULONG             BlockNum;                     //Image Block Number
      ULONG             BlockIndex;                   //Image Block Index

      //Camera control register groups
      CTRL_TRANSPORT    TransportCtrl;
      CTRL_IMAGE        ImageCtrl;
      CTRL_ACQ          AcqCtrl;
      CTRL_DIO          DioCtrl;
      CTRL_ANALOG       AnalogCtrl;
      CTRL_MISC         MiscCtrl;

} 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.

4.2 Header File GEV64MACROS.H

This header file defines structures required for handling GeniCam control registers

```
typedef struct _GEV_REGISTER
{
      ULONG addr;       //Register address
      ULONG size;       //Value size (typically 4)
      ULONG mask;       //Value Bit Mask
      TYPE64 value;     //Value (64-Bit union)

} GEV_REGISTER, *PGEV_REGISTER;
```

4.3 Debug Log File

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

Sample:

```
GEVCORE32 -> Sha32GevCreate
GEVCORE32 -> GetHostAddress
GEVCORE32 -> DrvCreate
GEVCORE32 -> CreateGevThread
GEVCORE32 -> CreateStationList
StationList.par
GEVCORE32 -> LoadStationList
StationList.par
GEVCORE32 -> LoadStationList
StationList.par
*** Frame Received ***
*** Discard ***
*** Frame Received ***
*** Discard ***
GEVCORE32 -> Sha32GevGetVersion
GEVCORE32 -> DrvGetVersion
GEVCORE32 -> Gev32EnableStation
Station1
GEVCORE32 -> Gev32ReadReg
GEVCORE32 -> GvcpCmd, Target IP: 192.168.1.5
GEVCORE32 -> WaitForGvcp [Enter], Target IP: 192.168.1.2
*** Frame Received ***
*** Synchronize ***
GEVCORE32 -> WaitForGvcp [Exit]
GEVCORE32 -> Gev32WriteReg
GEVCORE32 -> GvcpCmd, Target IP: 192.168.1.5
GEVCORE32 -> WaitForGvcp [Enter], Target IP: 192.168.1.2
*** Frame Received ***
*** Synchronize ***
GEVCORE32 -> WaitForGvcp [Exit]
-----------------------------------------------
Writing Station Station1 [HEARTBEAT_TIMEOUT] Addr:0x00000938 Value:0x00000bb8
GEVCORE32 -> Gev32WriteReg
GEVCORE32 -> GvcpCmd, Target IP: 192.168.1.5
GEVCORE32 -> WaitForGvcp [Enter], Target IP: 192.168.1.2
*** Frame Received ***
*** Synchronize ***
GEVCORE32 -> WaitForGvcp [Exit]
*** Writing OK ***
…
```

5 GigE-Vision Library Interface

The header file SHA64GEVCORE.H defines all required prototypes and parameters of the vision core library. The header file is based on the files GEV64COREDEF.H and ETH64COREDEF.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:

5.1 Basic Functions

5.1.1 Sha64GevCreate

This function initializes the GigE-Vision module states. On success the returning value is ERROR_SUCCESS, otherwise the returning value corresponds to that with GetLastError().

VC
```
ULONG Sha64GevCreate (PGEV_PARAMS);
```

Note: if no configuration file is given, the system will scan the bus automatically

5.1.2 Sha64GevDestroy

This function closes the GigE-Vision communication.

VC
```
ULONG Sha64GevDestroy(PGEV_PARAMS);
```

5.1.3 Sha64GevGetVersion

This function retrieves the version information of the GigE-Vision Master Library, the Ethernet Core Library, the Ethernet Core Driver, the SHA Dll, the SHA Library and the SHA Driver.

VC
```
ULONG Sha64GevGetVersion(PGEV_PARAMS);
```

Sample:

```
__pBlockListUser = (PUCHAR)malloc(IMG_BLOCK_SIZE * IMG_BLOCK_NUM);
{
      //Reset block memory
      memset(__pBlockListUser, 0, IMG_BLOCK_SIZE * IMG_BLOCK_NUM);

      //Required GEV parameters
      GEV_PARAMS GevParams;
      memset(&GevParams, 0, sizeof(GEV_PARAMS));
      GevParams.EthParams.dev_num = 0;
      GevParams.EthParams.period = REALTIME_PERIOD;
      GevParams.EthParams.fpAppTask = AppTask;
      strcpy(GevParams.szStationFile, "c:\\pnt\\StationList.par");

      //Enable GEV realtime core
      if (ERROR_SUCCESS == Sha32GevCreate(&GevParams))
      {
            //Init global elements
            __pUserStack      = GevParams.EthParams.pUserStack;
            __pSystemStack    = GevParams.EthParams.pSystemStack;
            __pUserList       = GevParams.pUserList;
            __pSystemList     = GevParams.pSystemList;
            __StationNum      = GevParams.StationNum;
            __fpGevEnter      = GevParams.fpGevEnter;
            __fpGevExit       = GevParams.fpGevExit;

            //Init block pool
            __pUserList[0].BlockList[0].pBuffer    = &__pBlockListUser[0];
            __pUserList[0].BlockList[0].BufferSize = IMG_BLOCK_SIZE;
            __pUserList[0].BlockList[1].pBuffer    =
                                    &__pBlockListUser[IMG_BLOCK_SIZE];
            __pUserList[0].BlockList[1].BufferSize = IMG_BLOCK_SIZE;
            __pUserList[0].BlockList[2].pBuffer    =
                                    &__pBlockListUser[IMG_BLOCK_SIZE*2];
            __pUserList[0].BlockList[2].BufferSize = IMG_BLOCK_SIZE;
            __pUserList[0].BlockNum   = IMG_BLOCK_NUM;
            __pUserList[0].BlockIndex = 0;

            //Check number of stations (Check Host IP address)
            if (__StationNum)
            {
                  //Enable stations
                  for (i=0; i<GevParams.StationNum; i++)
                        Gev32EnableStation(&__pUserList[i]);
      …
```

5.2 Command functions

The low level interface provides all function to handle GigE-Vision Commands

5.2.1 Set station name

VC `ULONG Gev64SetStationName(PSTATION_INFO pStation)`

5.2.2 Set station IP address

VC `ULONG Gev64SetStationIP(PSTATION_INFO pStation)`

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

} IP_PARAMS, *PIP_PARAMS;
```

Sample:

```
//Set station IP (optional)
pStation->Hdr.IpParams.ip_addr.bit8[0] = 192;
pStation->Hdr.IpParams.ip_addr.bit8[1] = 168;
pStation->Hdr.IpParams.ip_addr.bit8[2] = 1;
pStation->Hdr.IpParams.ip_addr.bit8[3] = 7;
Gev32SetStationIP(pStation);

//Set station name (optional)
strcpy(pStation->Hdr.szName, "Station1");
Gev32SetStationName(pStation);
```

5.2.3 Scan bus to station file

VC
```
ULONG Gev64ScanToStationFile(
                              PSTATION_INFO pStation, //optional
                              PULONG pStationNum);    //optional
```

5.2.4 Enable station

VC
```
ULONG Gev64EnableStation(PSTATION_INFO pStation);
```

5.2.5 Disable station

VC
```
ULONG Gev64DisableStation(PSTATION_INFO pStation);
```

5.3 GVCP Functions

The low level interface provides all function to handle the GigE-Vision GVCP protocol

5.3.1 GVCP Discovery

VC
```
ULONG Gev64Discovery(
                        PSTATION_INFO pStationList,   //Out: station list
                        PULONG pStationNum,           //Out: number of stations
                        BOOLEAN bBroadcast)           //In: broadcast discovery
```

5.3.2 GVCP ForceIP

VC
```
ULONG Gev64Forceip(
                        PSTATION_INFO pStation, //In: station pointer
                        BOOLEAN bWait)          //In: wait for answer
```

5.3.3 GVCP ReadMem

VC
```
ULONG SHAAPI Gev64ReadMem(
                        PSTATION_INFO pStation, //In: station pointer
                        ULONG MemSize,          //In: memory range
                        ULONG MemAddr,          //In: memory address
                        PUCHAR pMemData,        //Out: buffer
                        BOOLEAN bWait)          //In: wait for answer
```

5.3.4 GVCP WriteMem

VC
```
ULONG SHAAPI Gev64WriteMem(
                        PSTATION_INFO pStation, //In: station pointer
                        ULONG MemSize,          //In: memory range
                        ULONG MemAddr,          //In: memory address
                        PUCHAR pMemData,        //In: buffer
                        BOOLEAN bWait)          //In: wait for answer
```

5.3.5 GVCP ReadReg

```
VC ULONG SHAAPI Gev64ReadReg(
                              PSTATION_INFO pStation, //In: station pointer
                              PULONG pRegAddrList,    //In: register address list
                              PULONG pRegDataList,    //Out: register data list
                              ULONG RegNum,           //In: number of registers
                              BOOLEAN bWait)          //In: wait for answer
```

5.3.6 GVCP WriteReg

```
VC ULONG SHAAPI Gev64WriteReg(
                              PSTATION_INFO pStation, //In: station pointer
                              PULONG pRegAddrList,    //In: register address list
                              PULONG pRegDataList,    //In: register data list
                              ULONG RegNum,           //In: number of registers
                              BOOLEAN bWait)          //In: wait for answer
```

5.3.7 GVCP Macros

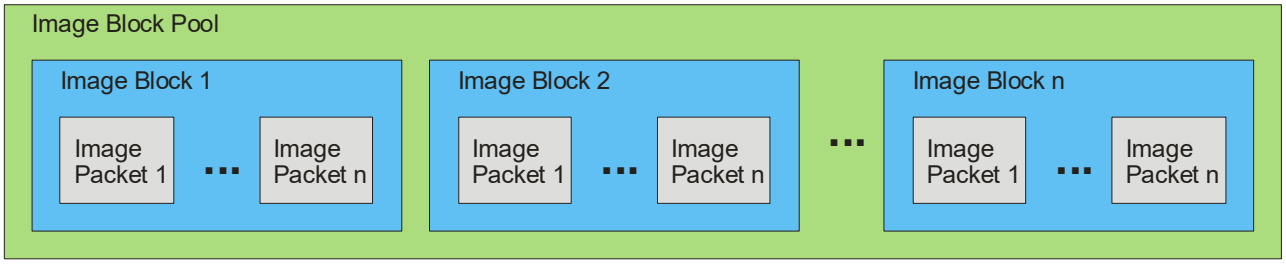
```
//Inline function to read from a single register
VC __inline ULONG __Gev32ReadSingleReg(
                              PSTATION_INFO pStation, //In: station pointer
                              ULONG RegAddr,          //In: register address
                              PULONG pRegData,        //Out: register data
                              BOOLEAN bWait = TRUE)   //In: wait for answer


//Inline function to write to a single register
VC __inline ULONG __Gev32WriteSingleReg(
                              PSTATION_INFO pStation, //In: station pointer
                              ULONG RegAddr,          //In: register address
                              ULONG RegData,          //In: register data
                              BOOLEAN bWait = TRUE)   //In: wait for answer
```

Sample:

```
ULONG SizeXMax = 0;
ULONG SizeYMax = 0;
ULONG Error = 0;

//Read Image max .Width
Error = __Gev32ReadSingleReg(pStation,  0x00030368, &SizeXMax);

//Read Image max. Height
Error = __Gev32ReadSingleReg(pStation,  0x00030388, &SizeYMax);


//Write image width
Error = __Gev32WriteSingleReg(pStation, 0x00030204,  SizeXMax);

//Write image height
Error = __Gev32WriteSingleReg(pStation, 0x00030224,  SizeYMax);
```

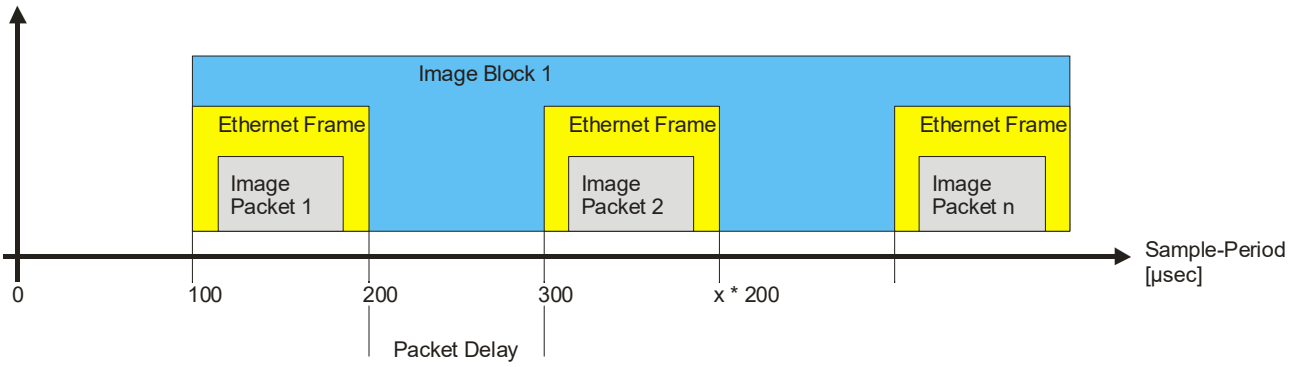
6 Image Buffer Management

The GigE Vision realtime core uses a multiple buffer management pool. This allows the decoupling of image acquisition and image processing (e.g. double buffering). Therefore memory for the buffer pool has to be allocated.



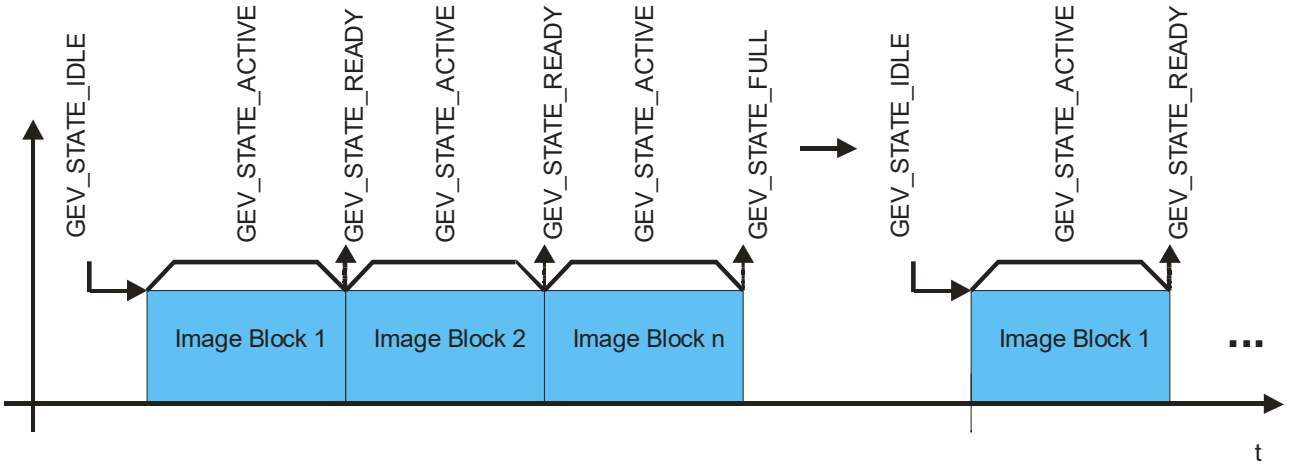
6.1 Image Block Timing

The GigE Vision realtime core fills this pool at realtime in a deterministic time grid using realtime samples (typically 100 µsec period) gathering all image packets to an image block. Therefore the control of the packet delay register is of importance. The packet delay is calculated due to the Tick-Frequency of the camera.



6.2 Image State Machine

The realtime core changes on each filled image block the state to GEV_STATE_READY. If all blocks have been filled the state changes to GEV_STATE_STOP. The user realtime task now changes this state back to GEV_STATE_IDLE to restart the cycle.



Sample:

```
#define IMG_BLOCK_SIZE        0x300000
#define IMG_BLOCK_NUM         3

HANDLE      __hBlockList = NULL;
PUCHAR      __pBlockListUser = NULL;
PUCHAR      __pBlockListSystem = NULL;

//Allocate memory device for image block queue
if (ERROR_SUCCESS == Sha32AllocMemWithTag(
            TRUE,                                   //Cached memory
            0,                                      //Memory tag
            IMG_BLOCK_SIZE * IMG_BLOCK_NUM,         //Memory size
            (PVOID*)&__pBlockListUser,              //Pointer to memory
                                                    //for Windows access
            (PVOID*)&__pBlockListSystem,            //Pointer to memory
                                                    //for Realtime access
            NULL,                                   //Physical memory address
            &__hBlockList))                         //Handle to memory device
      {
            //Reset block memory
            memset(__pBlockListUser, 0, IMG_BLOCK_SIZE * IMG_BLOCK_NUM);

            //Init block pool
            __pUserList[0].BlockList[0].pBuffer    = &__pBlockListUser[0];
            __pUserList[0].BlockList[0].BufferSize = IMG_BLOCK_SIZE;
            __pUserList[0].BlockList[1].pBuffer    =
                                        &__pBlockListUser[IMG_BLOCK_SIZE];
            __pUserList[0].BlockList[1].BufferSize = IMG_BLOCK_SIZE;
            __pUserList[0].BlockList[2].pBuffer    =
                                        &__pBlockListUser[IMG_BLOCK_SIZE*2];
            __pUserList[0].BlockList[2].BufferSize = IMG_BLOCK_SIZE;
            __pUserList[0].BlockNum   = IMG_BLOCK_NUM;
            __pUserList[0].BlockIndex = 0;
```

7 Realtime Operation

After enabling the GigE-Vision system (*Sha*64*GevCreate*) with a corresponding station list, the realtime tasks become active. The application realtime task is decorated by Realtime Wrapper functions:

```
//Call GEV enter function (Return: pointer to current station)
PSTATION_INFO pStation = __fpGevEnter(
                              __pSystemStack,   //In: Ethernet Stack Pointer
                              __pSystemList,    //In: Station List
                              __StationNum);    //In: Number of stations


typedef PSTATION_INFO   (__cdecl *FP_GEV_ENTER)
                                 (PETH_STACK, PSTATION_INFO, ULONG);
typedef VOID            (__cdecl *FP_GEV_EXIT) (PSTATION_INFO);
```

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

```
//Define GEV Wrapper States
enum _GEV_STATES
{
      GEV_STATE_STOP = 0,     // 0
      GEV_STATE_IDLE,         // 1
      GEV_STATE_ACTIVE,       // 2
      GEV_STATE_READY,        // 3
      GEV_STATE_FULL,         // 4
      GEV_STATE_ERROR         // 5
};
```

Get station state

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

Sample

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

      //*************************************
      //Call GEV enter function
      PSTATION_INFO pStation = __fpGevEnter(
                                    __pSystemStack,
                                    __pSystemList,
                                    __StationNum);
      if (pStation)
      //*************************************
      {
            if  (pStation->State == GEV_STATE_ACTIVE) { __PacketCnt++; }
            if ((pStation->State == GEV_STATE_READY) ||
                (pStation->State == GEV_STATE_FULL))
            {
                  //*******************************
                  //ToDo: block handling
                  //*******************************

                  //Increase image counter
                  __ImageCnt++;

                  __PacketRate = __PacketCnt;
                  __CycleRate  = __CycleCnt;

                  __CycleCnt = 0;
                  __PacketCnt = 0;

                  //Reset buffer  acquisition
                  pStation->State = GEV_STATE_IDLE;
            }
      }
      //*************************************
      //Call GEV exit function
      __fpGevExit(pStation);
      //*************************************

      //Increase period and cycle counter
      __PeriodCnt++;
      __CycleCnt++;
}
```

8 Error Handling

The master library provides an error handling and tracing mechanism.

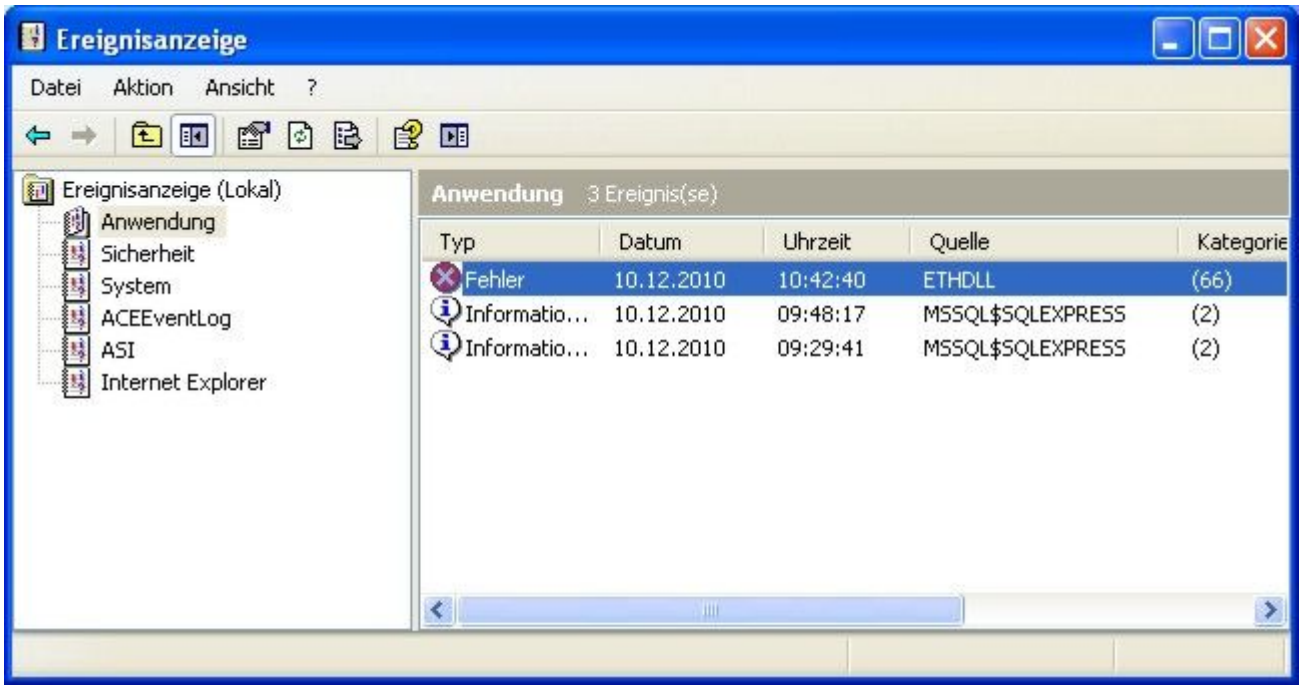
8.1 Debug LOG File

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

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

8.2 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.



9 Related Dokuments

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