Driver Manual

netANALYZER API

Windows XP/Vista/7/8

V1.5.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC091113DRV09EN | Revision 9 | English | 2016-11 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This manual describes the netANALYZER API, which are functions provided by the `netANALYZER_API.dll/netANALYZER_API.lib`.

1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
1	2009-11-28	MTr	all	Created for V1.3
2	2010-02-01	MTr/RG	all	Expanded
3	2010-02-01	MTr/RG		Examples added
4	2012-03-27	SD/HP/RG		Update for V1.4/GbE extension
5	2012-05-04	SD/HP/RG	3.1.5, 3.2.3 3.3.17.	Small corrections
6	2013-11-06	SD	3.1.8 3.1.9 3.1.10 3.1.11 3.1.12 3.3.14 3.3.15 3.3.21	Update for driver V1.5.3
7	2014-04-29	SD RG	3.3.19 3.3.19	Added note of callback restrictions Description of high load mode and small corrections added
8	2016-01-26	RG	3.3.14 3.3.15	Corrected function definitions for `netana_set_gpio_voltage` and `netana_get_gpio_voltage`
9	2016-11-04	RG	3.1.5 3.3.13 3.3.19	Small corrections

Table 1: List of Revisions

1.3 Terms, Abbreviations and Definitions

Term	Description
DPM	Dual Ported Memory
DMA	Direct Memory Access
GbE	Gigabit Ethernet
GPIO	General Purpose Input/Output

Table 2: Terms, Abbreviations and Definitions

1.4 Legal Notes

1.4.1 Copyright

1.4.2 Important Notes

The user manual, accompanying texts and the documentation were created for the use of the products by qualified experts, however, errors cannot be ruled out. For this reason, no guarantee can be made and neither juristic responsibility for erroneous information nor any liability can be assumed. Descriptions, accompanying texts and documentation included in the user manual do not present a guarantee nor any information about proper use as stipulated in the contract or a warranted feature. It cannot be ruled out that the user manual, the accompanying texts and the documentation do not correspond exactly to the described features, standards or other data of the delivered product. No warranty or guarantee regarding the correctness or accuracy of the information is assumed.

We reserve the right to change our products and their specification as well as related user manuals, accompanying texts and documentation at all times and without advance notice, without obligation to report the change. Changes will be included in future manuals and do not constitute any obligations. There is no entitlement to revisions of delivered documents. The manual delivered with the product applies.

Hilscher Gesellschaft für Systemautomation mbH is not liable under any circumstances for direct, indirect, incidental or follow-on damage or loss of earnings resulting from the use of the information contained in this publication.

1.4.3 Exclusion of Liability

The software was produced and tested with utmost care by Hilscher Gesellschaft für Systemautomation mbH and is made available as is. No warranty can be assumed for the performance and flawlessness of the software for all usage conditions and cases and for the results produced when utilized by the user. Liability for any damages that may result from the use of the hardware or software or related documents, is limited to cases of intent or grossly negligent violation of significant contractual obligations. Indemnity claims for the violation of significant contractual obligations are limited to damages that are foreseeable and typical for this type of contract.

It is strictly prohibited to use the software in the following areas:

- for military purposes or in weapon systems;
- for the design, construction, maintenance or operation of nuclear facilities;
- in air traffic control systems, air traffic or air traffic communication systems;
- in life support systems;
- in systems in which failures in the software could lead to personal injury or injuries leading to death.

We inform you that the software was not developed for use in dangerous environments requiring fail-proof control mechanisms. Use of the software in such an environment occurs at your own risk. No liability is assumed for damages or losses due to unauthorized use.

1.4.4 Export

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different counters, in particular those of Germany and the USA. The software may not be exported to countries where this is prohibited by the United States Export Administration Act and its additional provisions. You are obligated to comply with the regulations at your personal responsibility. We wish to inform you that you may require permission from state authorities to export, re-export or import the product.

2 Overview

This chapter gives a short overview how capturing with a netANALYZER card works internally. It explains capturing to file and the generic state machine for starting / stopping a capture.

The host needs to provide DMA buffers for the firmware running on the netANALYZER device. These buffers will be used by the firmware as a circular buffer. If a buffer is filled completely or a timeout occurs, this buffer will be released to the host. The host must now write these buffers to disk, or process them directly. After this step has been completed this buffer must be marked as free again so the firmware can continue using this buffer.

If the firmware does not find a usable buffer in the ring, it will abort capturing and indicate a capture error.

The following figure shows the capturing process:

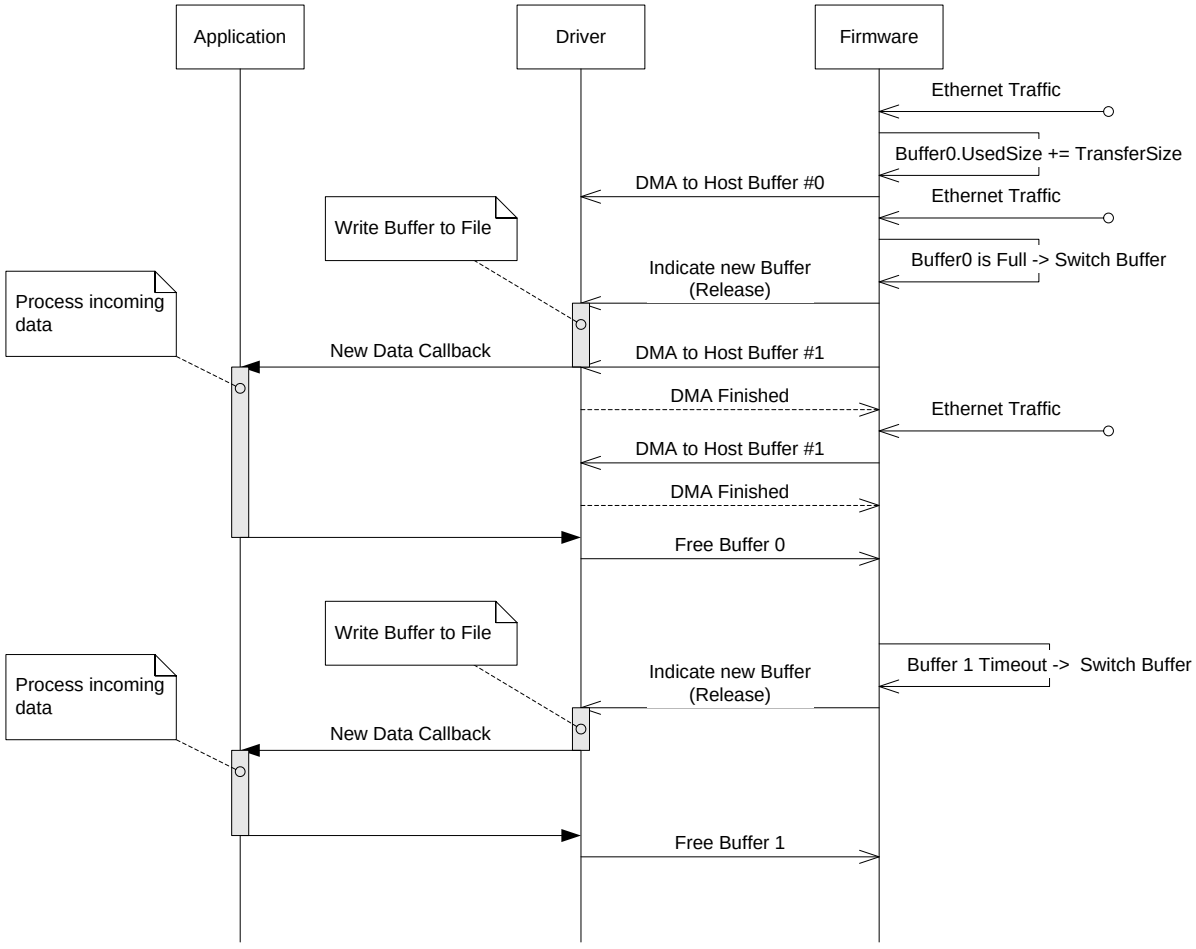

Figure 1: Overview - Capturing Process

2.1 Ring Buffer Mode

Since Version 1.3 the netANALYZER supports a ring buffer mode, which allows capturing to a set of files until a specific GPIO event occurs, or the user stops the capturing process.

The capture files are named with a running number appended to the base filename (see 3.3.10)

The following figure describes how the files are handled in ring buffer mode:

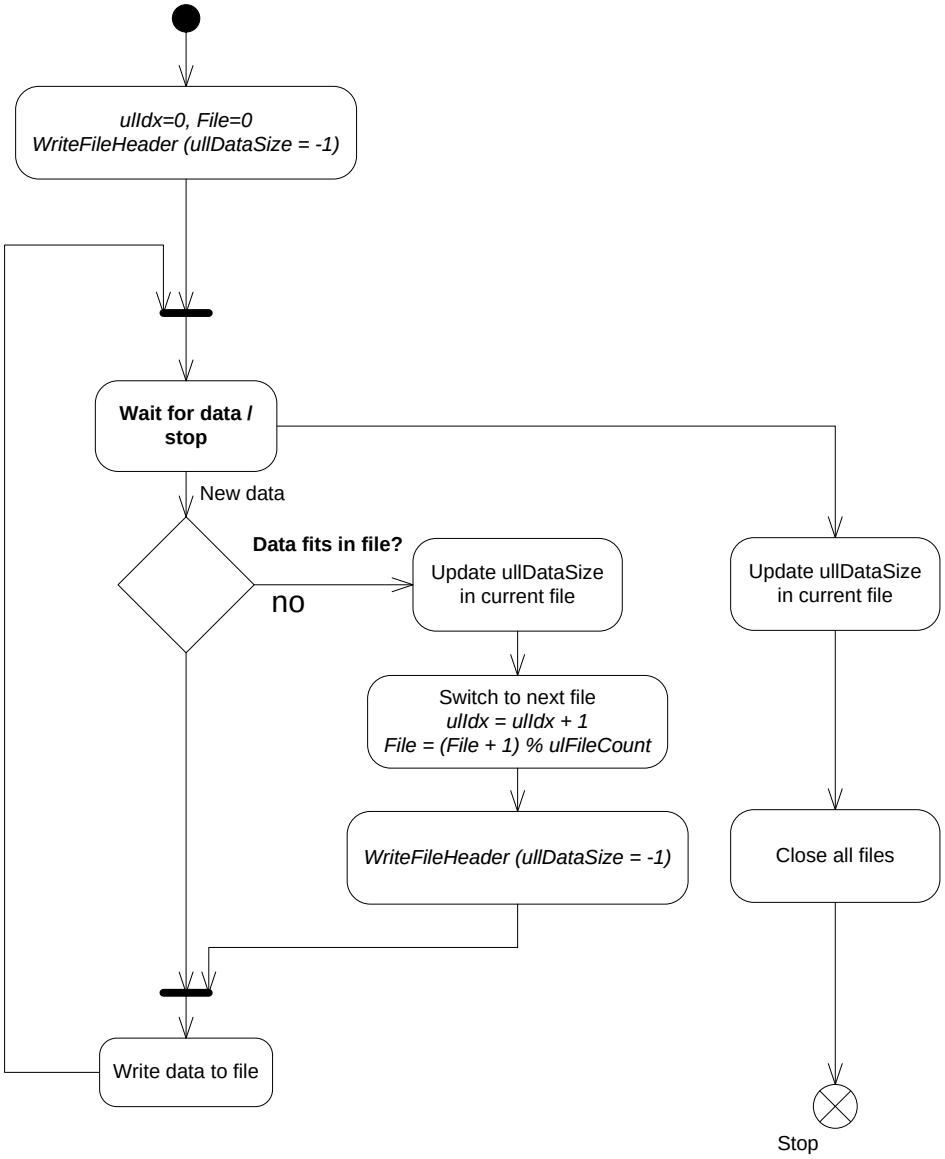

Figure 2: Ring Buffer Mode State Diagram

2.2 Capture File Structure

The capture files are stored in a proprietary format. The capturing file format described in the following applies for version 1.3 and higher of the netANALYZER API.

In contrast to prior versions of the netANALYZER API, capturing to multiple files is now supported and the file now contains a special 32 Byte file header. The data is stored in raw format starting after the header. These files always have a fixed size, independently of the length of captured data. This allows capturing in ring buffer mode.

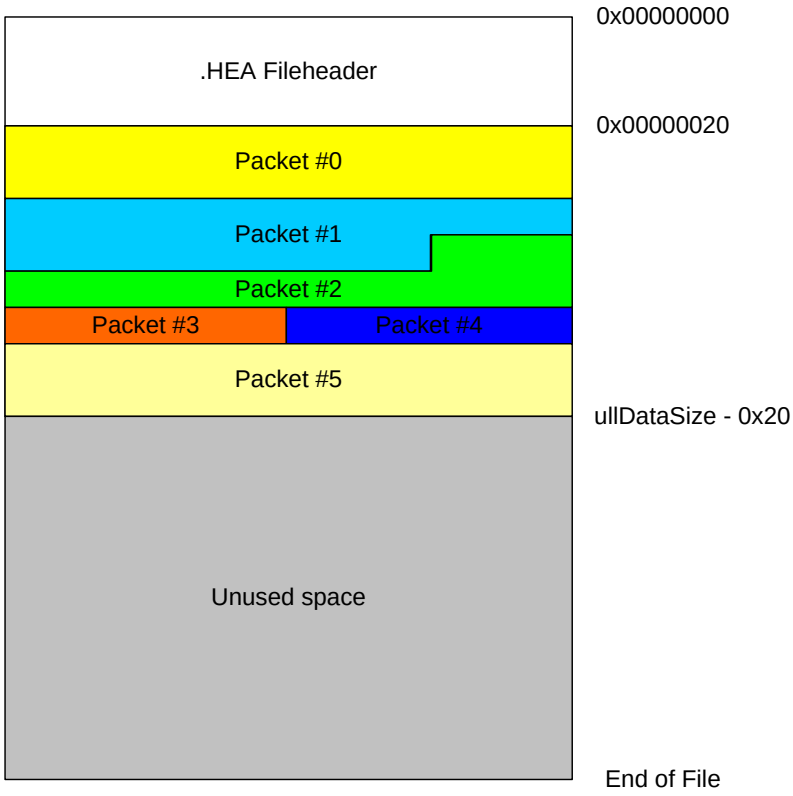

Figure 3: Capture File Overview - V1.3 and later

The .HEA Header has the following structure:

Element	Type	Description
ulCookie	UINT32	ASCII Cookie (.HEA) = 0x4145482E in little endian format
ulIdx	UINT32	Running file number, used for detecting file sequence in ring buffer mode
ullCaptureStart	UINT64	Capture start time. **Default:** Modified UNIX timestamp (nanoseconds since 1.1.1970)
ullDataSize	UINT64	Size of the captured data in this file. NETANA_FILE_HEADER_SIZE_INVALID (-1) indicates a not yet written / completed file.
lTimezoneCorrection	INT32	Time zone correction (to UTC) in seconds
ulReserved	UINT32	Reserved (set to zero)

Table 3: Structure of .HEA Header

The length of the data packets is aligned on 32 Bit boundaries packets. These are stored in the following format:

Element	Type	Description
ulHeader	UINT32	Packed information for the following Data: Bit 8 : 0=Ethernet Port, 1=GPIO Port Bit 9 : 0 = Ethernet mode, 1=Transparent mode Bits 10-13 : Version field, set it always to 1 Bits 14-15 : Port number this frame was received on Bits 16-27 : Length of the following data (Caplen)
ullTimestamp	UINT64	Nanosecond timestamp based on ullCaptureStart from header
abData	UINT8[Caplen]	Captured data. Size is included in ulHeader **Note:** This array will be padded up to the next 32 Bit boundary.

Table 4: Structure Capture File Data (V1.3 and later)

2.2.1.1 File sequence detection in ring buffer mode

To be able to process capture files from a ring buffered capture, it is necessary to detect all files belonging to the same capture.

The following steps need to be performed to read capture files, recorded in ring buffer mode:

1. Get ullCaptureStart from first file
2. Collect all files with the same value for ullCaptureStart
3. Search for the file with the lowest value in ulIdx and with a valid ullDataSize. --> This is the starting file of the capture
4. Iterate over all following files until the last file, or a file with an invalid DataSize, is reached

3 API Definitions

The netANALYZER API (located in the files `netANALYZER_API.dll/netANALYZER_API.lib`) offers several functions to access the device. These functions and their parameters / structures are defined in the following sections.

3.1 Structures

The following sections describe all structures that can be passed to the netANALYZER functions.

3.1.1 NETANA_FRAME_HEADER_T

This structure is the base of all captured data. It is used in the capture files (.HEA) and in the data callbacks.

Element	Type	Description
ulHeader	UINT32	Packed information for the following Data: Bit 0-7: Frame error code, see *Table 6: Frame Error Codes* below Bit 8 : 0=Ethernet Port, 1=GPIO Port Bit 9 : 0= Ethernet mode, 1=Transparent mode Bits 10-13 : Version field, set it always to 1 Bits 14-15 : Port number this frame was received on Bits 16-27 : Length of the following data (Caplen)
ullTimestamp	UINT64	Nanosecond timestamp based on ullReferencetime from netana_start_capture
abData	UINT8[Caplen]	Captured data. The size is included in ulHeader **Note:** This array will be padded up to the next 32 bit boundary.

Table 5: Structure: Capture Data Header

The following frame error codes are defined for bits 0 to 7 of ulHeader:

Frame error code NETANA_FRAME_HEADER_ERROR_CODE_...							
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
MSK_LONG_PREAMBLE	**MSK_SHORT_PREAMBLE**	**MSK_SHORT_FRAME**	**MSK_SFD_ERROR**	**MSK_TOO_LONG**	**MSK_FCS_ERROR**	**MSK_ALIGN_ERR**	**MSK_RX_ERR**
							MII RX_ER error
						Alignment error	
					FCS error		
				Frame too long			
			No valid SFD found				
		Frame smaller 64 bytes					
	Preamble shorter than 7 bytes						
Preamble longer than 7 bytes							

Table 6: Frame Error Codes

3.1.2 NETANA_DRIVER_INFORMATION_T

This structure is used in the "netana_device_information" (section NETANA_DEVICE_INFORMATION_T on page 13) to query all features / configuration of the driver.

Element	Type	Description
ulCardCnt	UINT32	Number of available netANALYZER devices
ulVersionMajor	UINT32	Driver version
ulVersionMinor	UINT32	
ulVersionBuild	UINT32	
ulVersionRevision	UINT32	
ulMaxFileCount	UINT32	Maximum number of files that can be created / handled.
ulDMABufferSize	UINT32	Size of DMA Buffers in Bytes
ulDMABufferCount	UINT32	Number of available DMA Buffers per Device
ulToolkitVersionMajor	UINT32	Toolkit version
ulToolkitVersionMinor	UINT32	
ulToolkitVersionBuild	UINT32	
ulToolkitVersionRevision	UINT32	
lMarshallerStatus	INT32	Error code status of the Marshaller-API interface
ulMarshallerVersionMajor	UINT32	Marshaller version (only valid if netAnaMarshaller-API extension is used)
ulMarshallerVersionMinor	UINT32	
ulMarshallerVersionBuild	UINT32	
ulMarshallerVersionRevision	UINT32	

Table 7: Structure: Driver Information

You can evaluate the error code status of the Marshaller-API interface `lMarshallerStatus` as follows:

- NETANA_NO_ERROR = 0 means the Marshaller is available and ready to operate.
- Otherwise an error code is returned in `lMarshallerStatus`. For meanings of Marshaller related error codes see *Table 8: Meanings of Marshaller related Error Codes* below.

Value	Definition	Description
0x80200008	NETANA_NO_ENTRY_FOUND	Wrong or missing registry entry
0x80200009	NETANA_FUNCTION_NOT_AVAILABLE	Missing function in the DLL of the Marshaller API
0x80220007	NETANA_FILE_OPEN_ERROR	Error when opening the Marshaller API specified in the reg path.

Table 8: Meanings of Marshaller related Error Codes

3.1.3 NETANA_DEVICE_INFORMATION_T

This structure defines all features of a netANALYZER device and is used in the following functions:

- `netana_enum_device` (see section *netana_enum_device* on page 25)
- `netana_device_info` (see section *netana_device_info* on page 27)

Element	Type	Description
szDeviceName	char[64]	Name of the device. (Used for netana_open_device)
ulPhysicalAddress	UINT32	Physical DPM Address of the device
ulDPMSize	UINT32	Size of the device DPM
ulUsedDPMSize	UINT32	Used DPM size of firmware
ulDPMLayoutVersion	UINT32	Version information of DPM Structure
ulInterrupt	UINT32	Interrupt Vector
ulDeviceNr	UINT32	Device number of netANALYZER card
ulSerialNr	UINT32	Device's serial number
ausHwOptions	UINT16[4]	Hardware options of all 4 ports
usManufacturer	UINT16	Manufacturer code of device
usProductionDate	UINT16	Production date
usDeviceClass	UINT16	Device class
bHwRevision	UINT8	Revision of the hardware
bHwCompatibility	UINT8	Hardware compatibility Index
ulOpenCnt	UINT32	Number of times this device has been opened.
ulPortCnt	UINT32	Number of available Ethernet ports
ulGpioCnt	UINT32	Number of available GPIO Inputs
ulFilterSize	UINT32	Size of the Ethernet filters in bytes
ulCounterSize	UINT32	Size of the statistics counter area in DPM
szFirmwareName	char[32]	Firmware name
ulVersionMajor	UINT32	Firmware Version
ulVersionMinor	UINT32	
ulVersionBuild	UINT32	
ulVersionRevision	UINT32	
ulExtendedInfoSize	UINT32	Size of the extended info field in bytes
ulExtendedInfoType	UINT32	Type of extended info
tExtendedInfo	NETANA_EXTENDED_DEV_INFO_T	Extended information block (depends on `ulExtendedInfoType`)

Table 9: Structure: Device Information

In order to be backward compatible with V1.3, it is possible to simply omit the last three elements of `NETANA_DEVICE_INFORMATION_T` (i.e. last three lines of *Table 9: Structure: Device Information*). In this case, the structure of V1.3 is used and the reduced length of the structure has to be taken into account in the calculation of parameter `ulDevInfoSize` in `netana_enum_device` or `netana_device_info`, respectively.

Otherwise, in case of referencing a remote device when calling the above mentioned functions,

- The full length of the structure NETANA_DEVICE_INFORMATION_T must be taken into account for the calculation of parameter ulDevInfoSize
- ulExtendedInfoSize is set to sizeof(NETANA_EXTENDED_DEV_INFO_T)+2*sizeof(UINT32).
- ulExtendedInfoType is set to the value 0x00476245 indicating ”GbE”.
- The extended information structure will be filled with additional information about the target system. The following table shows the information contained in the structure NETANA_EXTENDED_GBE_DEV_INFO_T within union NETANA_EXTENDED_DEV_INFO_T:

Element	Type	Description
abIpAddr	UINT8[4]	IP Address of the remote device
abSubnetMask	UINT8[4]	Subnet Mask of the remote device
abMacAddr	UINT8[6]	MAC Address of the remote device
ulServerVersionMajor	UINT32	Marshaller server version of the remote device
ulServerVersionMinor	UINT32	
ulServerVersionBuild	UINT32	
ulServerVersionRevision	UINT32	
ulDrvVersionMajor	UINT32	Driver version of the remote device
ulDrvVersionMinor	UINT32	
ulDrvVersionBuild	UINT32	
ulDrvVersionRevision	UINT32	
ulToolkitVersionMajor	UINT32	Toolkit version of the remote device
ulToolkitVersionMinor	UINT32	
ulToolkitVersionBuild	UINT32	
ulToolkitVersionRevision	UINT32	
ulOSVersionInfoSize	UINT32	OS version information of the remote device
ulOSMajorVersion	UINT32	
ulOSMinorVersion	UINT32	
ulOSBuildNumber	UINT32	
ulOSPlatformId	UINT32	
szCSDVersion	char[128]	

Table 10: Extended Device Information Structure: NETANA_EXTENDED_DEV_INFO_T

3.1.4 NETANA_FILTER_T

This structure defines a filter that can be used to filter out Ethernet traffic and is used in the following functions:

- `netana_get_filter` (see section *netana_get_filter* on page 33)
- `netana_set_filter` (see section netana_set_filter on page 38)

Element	Type	Description
ulFilterSize	UINT32	Size of the filter. **Note:** This value must match the length of the following pointers.
pbMask	UINT8*	Filter Mask (byte array of ulFilterSize)
pbValue	UINT8*	Filter Value (byte array of ulFilterSize)

Table 11: Structure: Filter Description

3.1.5 NETANA_GPIO_MODE_T

This structure defines the mode of a GPIO and is used in the following functions:

- `netana_get_gpio_mode` (see section *netana_get_gpio_mode* on page 32)
- `netana_set_gpio_mode` (see section *netana_set_gpio_mode* on page 32)

The following modes are available

Mode	Value	Action
NETANA_GPIO_MODE_NONE	0	don't capture GPIO
NETANA_GPIO_MODE_RISING_EDGE	1	Capture rising edge
NETANA_GPIO_MODE_FALLING_EDGE	2	Capture falling edge
NETANA_GPIO_MODE_OUTPUT	3	Set output mode, level is set by netana_set_gpio_output()
NETANA_GPIO_MODE_OUTPUT_PWM	4	Set output to one

Table 12: GPIO Modes

For the NETANA_GPIO_MODE_RISING_EDGE and NETANA_GPIO_MODE_FALLING_EDGE modes, use the following structure in order to define capturing / triggers:

Element	Type	Description
ulMode	UINT32	Mode for this GPIO. 0 : NETANA_GPIO_MODE_NONE – Don't use GPIO 1 : NETANA_GPIO_MODE_RISING_EDGE – Capture rising edges on this GPIO 2 : NETANA_GPIO_MODE_FALLING_EDGE – Capture falling edges on this GPIO
ulCaptureTriggers	UINT32	Setup trigger events (Bitmask) 0x00000001 : NETANA_GPIO_TRIGGER_START – Use first detected rising / falling edge to start capture 0x00010000 : NETANA_GPIO_TRIGGER_STOP – Use first detected rising / falling edge to stop capture
ulEndDelay	UINT32	Delay in multiples of 10 ns after capturing is stopped. **Note:** This value will be used if NETANA_GPIO_TRIGGER_STOP is set for this GPIO.

Table 13: Structure: GPIO Description

For the mode NETANA_GPIO_MODE_OUTPUT_PWM, use the following structure:

Element	Type	Description
ulMode	UINT32	Mode for this GPIO. 4 : NETANA_GPIO_MODE_OUTPUT_PWM - Sets GPIO to PWM mode
ulHiPeriod	UINT32	High period of signal generator in 10 ns. Minimum value is 1, maximum 100.000.000
ulLoPeriod	UINT32	Low period of signal generator in 10 ns. Min 1, maximum 100.000.000

Table 14: Structure: GPIO Description

PWM generation is only active while capture is started.

3.1.6 NETANA_PORT_STATE_T

Statistics and error counters for a specific port, used in the "netana_get_portstat" (see section *netana_get_portstat* on page 28) function.

Element	Type	Description
ulLinkState	UINT32	Current link state bitmask 0x00000001 : NETANA_LINK_STATE_UP – Link detected 0x00000002 : NETANA_LINK_STATE_SPEED100 – Link with 100 MBit detected (10MBit if bit is not set) 0x00000004: NETANA_LINK_STATE_SPEED_VALID – If set, the contents of NETANA_LINK_STATE_SPEED100 is valid. If not set, the contents of NETANA_LINK_STATE_SPEED100 is not reliable.
ullFrameReceivedOk	UINT64	Total number of successfully received Ethernet frames
ullRXErrors	UINT64	Number of receive errors
ullAlignmentErrors	UINT64	Number of frames with alignment errors (1 additional nibble received)
ullFrameCheckSequenceErrors	UINT64	Number of frames with a bad FCS (including short frames with a bad FCS)
ullFrameTooLongErrors	UINT64	Number of long frames received (>1514 bytes)
ullSFDErrors	UINT64	Number of Ethernet frames with a SFD (Start of frame delimiter) errors
ullShortFrames	UINT64	Short Frames (<60 Bytes)
ullFramesRejected	UINT64	Number of frames, that have been filtered out
ullLongPreambleCnt	UINT64	Frames with long preamble
ullShortPreambleCnt	UINT64	Frames with short preamble
ullBytesLineBusy	UINT64	Number of received bytes
ulMinIFG	UINT32	Lowest IFG (Interframe Gap) in 10 ns increments
ullTime	UINT64	Actual capture time as a running nanosecond counter (based on ullReferencetime passed to netana_start_capture)

Table 15: Structure: Portstate Description

3.1.7 NETANA_FILE_INFORMATION_T

This structure provides file capture statistics for currently running capture used in the "netana_file_info" (see section

netana_file_info on page 39) function.

Element	Type	Description
ulActualFileNr	UINT32	Current capture file
ulMaxFileCnt	UINT32	Maximum number of files in list
ullTotalBytesWritten	UINT64	Total number of bytes stored in files
ullMaxBytes	UINT64	Maximum number of bytes that can be stored
ulRingBufferMode	UINT32	!=0 if capturing is done in Ring buffermode

Table 16: Structure: File Information Description

3.1.8 NETANA_MNGMT_DEV_SCAN_IN_T

This structure is required when calling the netana_mngmt_exec_cmd() function with the Management Command NETANA_MNGMT_CMD_DEV_SCAN.

Element	Type	Description
pfnCallBack	PFN_SCAN_CALLBACK	Function pointer to scan callback. For more information see PFN_SCAN_CALLBACK on page 22.

Table 17: Structure: NETANA_MNGMT_DEV_SCAN_IN_T Description

3.1.9 NETANA_MNGMT_DEV_SCAN_OUT_T

This structure is required when calling the netana_mngmt_exec_cmd() function with the Management Command NETANA_MNGMT_CMD_DEV_SCAN.

Element	Type	Description
ulNofDevices	UINT32	Number of found devices

Table 18: Structure: NETANA_MNGMT_DEV_SCAN_OUT_T Description

3.1.10 NETANA_MNGMT_GET_DEV_FEATURE_IN_T

This structure is required when calling the netana_mngmt_exec_cmd() function with the Management Command NETANA_MNGMT_CMD_GET_DEV_FEATURE.

Element	Type	Description
hDev	NETANA_HANDLE	Handle of the requested device

Table 19: Structure: NETANA_MNGMT_GET_DEV_FEATURE_IN_T Description

3.1.11 NETANA_MNGMT_GET_DEV_FEATURE_OUT_T

This structure provides the device features and limitations. The information structure is returned when calling the netana_mngmt_exec_cmd() function with the Management Command NETANA_MNGMT_CMD_GET_DEV_FEATURE.

Element	Type	Description
ulStructVersion	UINT32	Version of this structure (currently 0x00000000)
ulPhysType	UINT32	Type of supported physics 0x00000000: Ethernet
ulNumPhysPorts	UINT32	Number of available Physics ports
ulPhysTapPresent	UINT32	Physics TAP is built in (0: no, 1: yes) Bitmask, each represents one Eth port from 0..max.31 (currently up to 4)
ulPhysForwardingSupport	UINT32	Support of Physics frame forwarding if no TAP is present. (0: no, 1: yes) Bitmask, each represents one Eth port from 0..max.31 (currently up to 4)
ulPhysPortSpeedSupport	UINT32	Bitmask, each represents one Physics port from 0..max.31 (currently up to 4) 0: does not support indicating the link speed 1: does support indicating the link speed
ulPhysTransparentModeSupport	UINT32	Bitmask, each represents one Physics port from 0..max.31 (currently up to 4) 0: does not support capturing in transparent mode 1: does support capturing in transparent mode
ulNumGpios	UINT32	Number of available GPIOs
ulGpioInputRisingSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support input mode to capture rising edge 1: does support input mode to capture rising edge
ulGpioInputFallingSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support input mode to capture falling edge 1: does support input mode to capture falling edge
ulGpioOutputModeSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support output mode 1: does support output mode
ulGpioOutputPWMSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support output PWM 1: does support output PWM
ulGpioSyncInSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support sync input at GPIO 1: does support sync input at GPIO
ulGpioTriggerStartSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support trigger start event at GPIO 1: does support trigger start event at GPIO
ulGpioTriggerStopSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up

		to 4) 0: does not support trigger stop event at GPIO 1: does support trigger stop event at GPIO
ulGpioVoltage3VSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support 3.3V at GPIO 1: does support 3.3V at GPIO
ulGpioVoltage24VSupport	UINT32	Bitmask, each represents one GPIO from 0..max.31 (currently up to 4) 0: does not support 24V at GPIO 1: does support 24V at GPIO
ulSyncSupport	UINT32	The device offers time sync support 0: no 1: yes, via DPM

Table 20: Structure: NETANA_MNGMT_GET_DEV_FEATURE_OUT_T Description

3.1.12 NETANA_MNGMT_SET_DEV_CLASS_FILTER_IN_T

This structure is required when calling the netana_mngmt_exec_cmd() function with the Management Command NETANA_MNGMT_CMD_SET_DEV_CLASS_FILTER.

Element	Type	Description
ulDeviceClass	UINT32	Device class which should be applied. Mask need to be set to the device class to be filtered. NETANA_DEV_CLASS_NANL_500 = netANALYZER devices NETANA_DEV_CLASS_NSCP_100 = netSCOPE devices

Table 21: Structure: NETANA_MNGMT_SET_DEV_CLASS_FILTER_IN_T Description

3.2 Callbacks

The netANALYZER API can notify the user for different events via callbacks. This chapter describes the available callbacks.

3.2.1 PFN_STATUS_CALLBACK

This function is called, if the device changes its state during capture.

Note: Note the callback restrictions under netana_start_capture on page 40.

Function definition:

```
void StatusCallback(UINT32 ulCaptureState, UINT32 ulCaptureError, void* pvUser)
```

Parameter	Description
ulCaptureState	State of the capturing 0: NETANA_CAPTURE_STATE_OFF – Capture is inactive 1: NETANA_CAPTURE_STATE_START_PENDING – Firmware is preparing to start capturing 2: NETANA_CAPTURE_STATE_RUNNING – Capture is active 3: NETANA_CAPTURE_STATE_STOP_PENDING – Capturing is stopped and firmware waits for user to call netana_stop_capture. The error is indicated in the ulCaptureError parameter
ulCaptureError	Error indication of firmware: 0x00000000: NETANA_CAPTURE_ERROR_STOP_TRIGGER – Capturing stopped by user or GPIO trigger 0xC0660004: NETANA_CAPTURE_ERROR_NO_DMACHANNEL – Firmware was unable to find a free DMA channel 0xC0660005: NETANA_CAPTURE_ERROR_URX_OVERFLOW – Critical error in XPEC during reception 0xC066000B: NETANA_CAPTURE_ERROR_NO_HOSTBUFFER – No free host buffer found. This indicates the filesystem for filewriting is slow or the application takes to long handling the incoming data. 0xC066000C: NETANA_CAPTURE_ERROR_NO_INTRAM – Firmware is out of memory resources and is unable to buffer more data. This may also be caused by a slow file system or a slow application 0xC066000D: NETANA_CAPTURE_ERROR_FIFO_FULL – Firmware is out of FIFO resources and is unable to buffer more data. This may also be caused by a slow file system or a slow application 0xC0770000: NETANA_CAPTURE_ERROR_DRIVER_FILE_FULL – End of capture file reached. Driver has stopped capturing. 0xC0770001: NETANA_CAPTURE_ERROR_DRIVER_INVALID_BUFFERSIZE - Internal capture buffer overflow (no free INTRAM)
pvUser	User specific parameter passed to netana_start_capture

Table 22: Status Callback - Parameters

3.2.2 PFN_DATA_CALLBACK

This function is called, if the device has captured new data.

Note: Note the callback restrictions under netana_start_capture on page 40.

Function definition:

```
void DataCallback(void* pvData, UINT32 ulDataLen, void* pvUser)
```

Parameter	Description
pvData	Pointer to captured data. For structure of the data, refer to section *NETANA_FRAME_HEADER_T* on page 11.
ulDataLen	Length of data in buffer
pvUser	User specific parameter passed to netana_start_capture

Table 23: Data Callback - Parameters

3.2.3 PFN_SCAN_CALLBACK

This function is called cyclically during bus scan. At a “local” device scan, *bProgress* is set to 0.

Function definition:

```
void ScanCallback(UINT8 bProgress, UINT32 ulFoundNumber, char* szLastFound, void* pvUser)
```

Parameter	Description
bProgress	Progress of bus scan in percent (0-100).
ulFoundNumber	Number of found devices.
szLastFound	Name of the last found device
pvUser	User specific parameter passed to ScanCallback()

Table 24: Scan Callback - Parameters

3.3 Functions

The netANALYZER function offers driver based functions to find, open / close devices on the computer and device specific functions to configure devices and start / stop Ethernet capturing.

The following functions are available:

Function	Description
Driver based	
netana_driver_information	Get driver information (Version, number of devices)
netana_get_error_description	Get error description text (english) from an error code
netana_enum_device	Enumerate found netANALYZER devices
netana_open_device	Exclusively open a netANALYZER device
netana_close_device	Close an exclusively opened netANALYZER device
Device based	
netana_device_info	Get device information (Firmware version, supported features, etc.)
netana_get_portstat	Get error counters, link state and statistics counters.
netana_get_state	Get current capturing state
netana_access_phy_reg	Read/Write PHY registers on a specific port
netana_set_filelist	De-/Activate capturing traffic to file for next capture
netana_get_gpio_mode	Get Configuration of a GPIO Input
netana_set_gpio_mode	Set Configuration of a GPIO Input
netana_set_gpio_output	Set Configuration of a GPIO Output
netana_set_gpio_voltage	Set voltage configuration of GPIOs
netana_get_gpio_voltage	Get voltage configuration of a GPIO
netana_get_filter	Get active filters for a given port
netana_set_filter	Set active filters for a given port
netana_file_info	Get current file capture state (Actual file written, total bytes captured, etc.)
netana_start_capture	Start a live capture
netana_stop_capture	Stop a running live capture
netana_mngmt_exec_cmd	General function interface

3.3.1 netana_driver_information

Query driver specific information.

```
INT32 netana_driver_information(     UINT32 ulDrvInfoSize,
     NETANA_DRIVER_INFORMATION_T* ptDrvInfo);
```

Parameter	Description
ulDrvInfoSize	Size of the structure passed in ptDrvInfo
ptDrvInfo	Returned driver information

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_DEVICE_NOT_FOUND (0x80220003)	Device has not been found
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter, occurs when `ptDrvInfo == NULL`

3.3.2 netana_get_error_description

Get the error description text for a return code from the netANALYZER API.

```
INT32 netana_get_error_description(  INT32 lError,
     UINT32 ulBufferSize,
     char* szBuffer);
```

Parameter	Description
lError	Error code to return description for
ulBufferSize	Size of the passed buffer
szBuffer	Buffer for returned text

Returns:

NETANA_NO_ERROR (0x0)	Success

3.3.3 netana_enum_device

Enumerate available devices-

```
INT32 netana_enum_device( UINT32 ulCardNr,
     UINT32 ulDevInfoSize,
     NETANA_DEVICE_INFORMATION_T* ptDevInfo);
```

Parameter	Description
ulCardNr	Number of the device to return
ulDevInfoSize	Size of the structure passed in ptDevInfo
ptDevInfo	Returned device information

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_DEVICE_NOT_FOUND (0x80220003)	Device with card number was not found
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL

Example:

```
UINT32                      ulCard   = 0;
NETANA_DEVICE_INFORMATION_T tDevInfo = {0};

while(NETANA_NO_ERROR (0x0) == netana_enum_device(ulCard++, sizeof(tDevInfo), &tDevInfo))
{
...
}
```

3.3.4 netana_open_device

Opens a device for exclusive access. A device can only be opened once even from the same process.

```
INT32 netana_open_device( char* szDevice,
     NETANA_HANDLE* phDev);
```

Parameter	Description
szDevice	Device description
phDev	Returned handle for device access

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (at least one parameter points to 0)
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL

Example:

```
NETANA_HANDLE hDev;

if(NETANA_NO_ERROR (0x0) == netana_open_device(“netANALYZER_0”, &hDev)
{
...
}
```

3.3.5 netana_close_device

Close a previously opened device.

```
INT32 netana_close_device(NETANA_HANDLE hDev);
```

Parameter	Description
hDev	Device returned from a previous call to netana_open_device

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL

3.3.6 netana_device_info

Get device information for an open device.

```
INT32 netana_device_info(  NETANA_HANDLE hDev,
     UINT32 ulDevInfoSize,
     NETANA_DEVICE_INFORMATION_T* ptDevInfo);
```

Parameter	Description
hDev	Device handle
ulDevInfoSize	Size of the structure passed in ptDevInfo
ptDevInfo	Returned device information

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (at least one parameter points to 0)

3.3.7 netana_get_portstat

Retrieve diagnostic / statistic counters for a specific port.

```
INT32 netana_get_portstat( NETANA_HANDLE hDev,
     UINT32 ulPort,
     UINT32 ulSize,
     NETANA_PORT_STATE_T* ptStatus);
```

Parameter	Description
hDev	Device handle
ulPort	Port to get information for
ulSize	Size of the structure passed in ptStatus
ptStatus	Returned port state

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong configuration parameter)
NETANA_FUNCTION_FAILED (0x80200004)	Function failed Problem at reading or setting the status

3.3.8 netana_get_state

Get current capturing state / error.

```
INT32 netana_get_state(   NETANA_HANDLE hDev,
     UINT32* pulCaptureState,
     UINT32* pulCaptureError);
```

Parameter	Description
hDev	Device handle
pulCaptureState	Returned capture state 0: NETANA_CAPTURE_STATE_OFF – Capture is inactive 1: NETANA_CAPTURE_STATE_START_PENDING – Firmware is preparing to start capturing 2: NETANA_CAPTURE_STATE_RUNNING – Capture is active 3: NETANA_CAPTURE_STATE_STOP_PENDING – Capturing is stopped and firmware waits for user to call netana_stop_capture.
pulCaptureError	Returned capture error (Valid if Capture state is NETANA_CAPTURE_STATE_STOP_PENDING)

Returns:

NETANA_NO_ERROR (0x0)	Success

3.3.9 netana_access_phy_reg

Read / Write PHY registers on a specific port.

```
INT32 netana_access_phy_reg(    NETANA_HANDLE hDev,
     UINT32 ulDirection,
     UINT8 bPhyNum,
     UINT8 bPhyReg,
     UINT16* pusValue,
     UINT32 ulTimeout);
```

Parameter	Description
hDev	Device handle
ulDirection	0: NETANA_PHY_DIRECTION_READ – Read from PHY 1: NETANA_PHY_DIRECTION_WRITE – Write to PHY
bPhyNum	Phy / Port number (0..3)
bPhyReg	Register to read / write (0..31)
pusValue	Write value if Direction is NETANA_PHY_DIRECTION_WRITE Returned value if Direction is NETANA_PHY_DIRECTION_READ
ulTimeout	Timeout in ms to wait for access to complete

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong configuration parameter)
NETANA_FUNCTION_FAILED (0x80200004)	Function failed Problem at reading or setting the status

3.3.10 netana_set_filelist

Set file data and activate capturing to .HEA-file.

Files will be named “<szPath>\<szBaseFilename>_<Filenumber>.hea”.

Example:

```
szPath=”C:\”
szBaseFileName=”capture”
ulFileCount=2
```

The following files will be created:

`C:\capture_0.hea`

`C:\capture_1.hea`

```
INT32 netana_set_filelist( NETANA_HANDLE hDev,
     char* szPath,
     char* szBaseFilename,
     UINT32 ulFileCount,
     UINT64 ullFileSize);
```

Parameter	Description
hDev	Device handle
szPath	Directory to place files in. **Note:** UNC paths are not supported.
szBaseFilename	Base filename without extension
ulFileCount	Number of files to create
ullFileSize	Size in Bytes of each file

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong configuration parameter)
NETANA_CAPTURE_ACTIVE (0x8022000B)	Capture is currently active

3.3.11 netana_get_gpio_mode

Get the current configuration of a GPIO.

```
INT32 netana_get_gpio_mode(     NETANA_HANDLE hDev,
     UINT32 ulGpio,
     NETANA_GPIO_MODE_T* ptMode);
```

Parameter	Description
hDev	Device handle
ulGPIO	Number of the GPIO
ptMode	Returned GPIO configuration

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong GPIO number)
NETANA_CAPTURE_NOT_ACTIVE (0x8022000C)	Capturing is currently stopped

3.3.12 netana_set_gpio_mode

Configure a GPIO pin as input / trigger.

```
INT32 netana_set_gpio_mode(     NETANA_HANDLE hDev,
     UINT32 ulGpio,
     NETANA_GPIO_MODE_T* ptMode);
```

Parameter	Description
hDev	Device handle
ulGPIO	Number of the GPIO mode
ptMode	GPIO configuration

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong GPIO number)
NETANA_CAPTURE_NOT_ACTIVE (0x8022000C)	Capturing is currently stopped
NETANA_FUNCTION_FAILED (0x80200004)	Function failed PWM is already active

3.3.13 netana_set_gpio_output

Configures a GPIO used as output.

```
INT32 netana_set_gpio_output(   NETANA_HANDLE hDev,
     UINT32 ulGpio,
     UINT32 ulLevel);
```

Parameter	Description
hDev	Device handle
ulGPIO	Number of the GPIO
ulLevel	Output level to be driven (0 / 1)

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong configuration parameter)
NETANA_FUNCTION_FAILED (0x80200004)	Function failed GPIO has not been configured for output

3.3.14 netana_set_gpio_voltage

Configures the GPIO voltage settings.

```
INT32 netana_set_gpio_voltage(  NETANA_HANDLE hDev,
     UINT32 ulGpioSelMsk,
     UINT32 ulGpioVoltageSel);
```

Parameter	Description
hDev	Device handle
ulGpioSelMask	Mask of the GPIOs to be affected by the voltage setting delivered by ulGpioVoltageSel. NETANA_GPIO0_VOLTAGE_MASK = GPIO0 NETANA_GPIO1_VOLTAGE_MASK = GPIO1 NETANA_GPIO2_VOLTAGE_MASK = GPIO2 NETANA_GPIO3_VOLTAGE_MASK = GPIO3
ulGpioVoltageSel	Voltage selection: NETANA_GPIO_VOLTAGE_3V = 3V NETANA_GPIO_VOLTAGE_24V = 24V

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong configuration parameter)

3.3.15 netana_get_gpio_voltage

The function returns the currently active GPIO voltage selection for a specific GPIO.

```
INT32 netana_get_gpio_voltage(  NETANA_HANDLE hDev,
     UINT32 ulGpio,
     UINT32* pulGpioVoltageSel);
```

Parameter	Description
hDev	Device handle
ulGPIO	Number of the GPIO
pulGpioVoltageSel	Pointer to a buffer in which the currently active voltage setting will be returned.

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter

3.3.16 netana_get_filter

Get the currently active filters of an Ethernet port.

```
INT32 APIENTRY netana_get_filter(    NETANA_HANDLE hDev,
     UINT32 ulPort,
     NETANA_FILTER_T* ptFilterA,
     NETANA_FILTER_T* ptFilterB,
     UINT32* pulRelationShip);
```

Parameter	Description
hDev	Device handle
ulPort	Ethernet port number
ptFilterA	Filter A. **Note:** Pointers in these structure must be allocated by user and the correct length must be passed.
ptFilterB	Filter B. **Note:** Pointers in these structure must be allocated by user and the correct length must be passed.
pulRelationShip	Relationship of both filters (Bitmask) 0x00000001 NETANA_FILTER_RELATION_A_OR_B 0 = Result = Filter A AND Filter B (corresponds to NETANA_FILTER_RELATION_A_AND_B = 0) 1 = Result = Filter A OR Filter B (corresponds to NETANA_FILTER_RELATION_A_OR_B = 1) 0x00000080 NETANA_FILTER_RELATION_REJECT_FILTER 0 = Accept frames if filter matches (corresponds to NETANA_FILTER_RELATION_ACCEPT_FILTER = 0) 1 = Reject frames if filter matches (corresponds to NETANA_FILTER_RELATION_REJECT_FILTER = 0x00000080) 0x00008000 NETANA_FILTER_RELATION_ACCEPT_ERROR_FRAMES 0 = Error frames are rejected 1 = Error frames are accepted 0x00010000 NETANA_FILTER_RELATION_FILTER_B_NOT 0 = Do not negate Filter B 1 = Negate Filter B 0x00800000 NETANA_FILTER_RELATION_FILTER_B_ENABLE 0 = Do not enable Filter B 1 = Enable Filter B 0x01000000NETANA_FILTER_RELATION_FILTER_A_NOT 0 = Do not negate Filter A 1 = Negate Filter A 0x80000000 NETANA_FILTER_RELATION_FILTER_A_ENABLE 0 = Do not enable Filter A 1 = Enable Filter A

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PORT (0x80200002)	Invalid port number

3.3.17 netana_set_filter

Set the capturing filters of an Ethernet port.

```
INT32 APIENTRY netana_set_filter(    NETANA_HANDLE hDev,
     UINT32 ulPort,
     NETANA_FILTER_T* ptFilterA,
     NETANA_FILTER_T* ptFilterB,
     UINT32 ulRelationShip);
```

Parameter	Description
hDev	Device handle
ulPort	Ethernet port number
ptFilterA	Filter A. **Note:** Pointers in these structure must be allocated by user and the correct length must be passed.
ptFilterB	Filter B. **Note:** Pointers in these structure must be allocated by user and the correct length must be passed.
ulRelationShip	Relationship of both filters (Bit mask) 0x00000001 NETANA_FILTER_RELATION_A_OR_B 0 = Result = Filter A AND Filter B (corresponds to NETANA_FILTER_RELATION_A_AND_B = 0) 1 = Result = Filter A OR Filter B (corresponds to NETANA_FILTER_RELATION_A_OR_B=1) 0x00000080 NETANA_FILTER_RELATION_REJECT_FILTER 0 = Accept frames if filter matches (corresponds to NETANA_FILTER_RELATION_ACCEPT_FILTER = 0) 1 = Reject frames if filter matches (corresponds to NETANA_FILTER_RELATION_REJECT_FILTER = 0x00000080) 0x00008000 NETANA_FILTER_RELATION_ACCEPT_ERROR_FRAMES 0 = Error frames are rejected 1 = Error frames are accepted 0x00010000 NETANA_FILTER_RELATION_FILTER_B_NOT 0 = Do not negate Filter B 1 = Negate Filter B 0x00800000 NETANA_FILTER_RELATION_FILTER_B_ENABLE 0 = Do not enable Filter B 1 = Enable Filter B 0x01000000 NETANA_FILTER_RELATION_FILTER_A_NOT 0 = Do not negate Filter A 1 = Negate Filter A 0x80000000 NETANA_FILTER_RELATION_FILTER_A_ENABLE 0 = Do not enable Filter A 1 = Enable Filter A

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PORT (0x80200002)	Invalid port number

3.3.18 netana_file_info

Get the current capture file state.

```
INT32 netana_file_info(   NETANA_HANDLE hDev,
     UINT32 ulFileInfoSize,
     NETANA_FILE_INFORMATION_T* ptFileInfo);
```

Parameter	Description
hDev	Device handle
ulFileInfoSize	Size of the passed structure in ptFileInfo
ptFileInfo	Returned File information

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (at least one parameter points to 0)
NETANA_CAPTURE_NOT_ACTIVE (0x8022000C)	Capturing is currently stopped
NETANA_FILECAPTURE_NOT_ACTIVE (0x8022000D)	Capturing to file is not enabled

3.3.19 netana_start_capture

Start a live capture. This function must be called after all ports have been correctly configured.

Note: Always call the function netana_set_filelist, before calling netana_start_capture, if capturing to file is intended.

```
INT32 netana_start_capture(     NETANA_HANDLE hDev,
     UINT32 ulCaptureMode,
     UINT32 ulPortEnableMask,
     UINT32 ulMacMode,
     UINT64 ullReferenceTime,
     PFN_STATUS_CALLBACK pfnStatus,
     PFN_DATA_CALLBACK pfnData,
     void* pvUser);
```

Parameter	Description
hDev	Device handle
ulCaptureMode	0: NETANA_CAPTUREMODE_DATA - Capture whole Ethernet frames 1: NETANA_CAPTUREMODE_TIMESTAMPS - Only capture timestamps (reception time) of incoming frames. 2: NETANA_CAPTUREMODE_HIGHLOAD - A special mode for measurements to capture reduced frames at very high network load. In order to avoid PCI bus overload, all frames are truncated to a length of 58 bytes! See description below! 0x80000000: NETANA_CAPTUREMODE_RINGBUFFER - Set this bit to enable data capturing in ringbuffer mode (only valid for capturing to file)
ulPortEnableMask	Bitmask of enabled Ethernet ports. Each bit represents a port.
ulMacMode	0: NETANA_MACMODE_ETHERNET - Capture IEEE 802.3 frames in Ethernet mode 1: NETANA_MACMODE_TRANSPARENT - Capture all bytes on Ethernet without regarding any specifications
ullReferenceTime	Capture reference time in nanoseconds. This value is expected to be the UNIX time (Seconds since 1.1.1970) in nanoseconds, as Wireshark expects this format. **Note:** When using a non-UNIX reference time, conversion with the standard tools will produce a wrong timestamp. When using customer tools, the reference time may by set to any customer value.
pfnStatus	Status change notification callback. Set to NULL if not needed ***when registering callback note the following restrictions***.
pfnData	Callback for captured data. Set to NULL if not needed. **Note:** If this parameter is NULL, a capturing file needs to be set via netana_set_filelist ***when registering callback note the following restrictions***.
pvUser	User parameter passed on callbacks

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter (wrong configuration parameter)
NETANA_MEMORY_MAPPING_FAILED (0x80220006L)	Error mapping memory to user application
NETANA_THREAD_CREATION_FAILED (0x8022000F)	Error creating worker thread
NETANA_CAPTURE_ACTIVE (0x8022000B)	Capture is already active
NETANA_FILE_WRITE_FAILED (0x8022000A)	Writing the capture file failed
NETANA_FUNCTION_FAILED (0x80200004)	Function failed Problem at reading or setting the status

Restrictions of registered callbacks

- **Restricted API usage within callback**

  It is not possible to call any other function of the netANALYZER API within the callbacks.

- **Duration of callback is limited**

  The callback functions registered via the parameter pfnStatus and pfnData must ensure not to block the execution. Since the driver will wait for the callbacks to be completed, during callback execution no further data or state changes will be processed. Therefore the average of the callback's execution time must not exceed the cycle time of a notifcation. Otherwise the DMA buffer resources will be exhausted and the running capture will fail with the error NETANA_CAPTURE_ERROR_NO_HOSTBUFFER.

  The period of the notification callbacks depends on

  A) The configured default DMA buffer timeout (default value 50 ms)

  Example:
  Default timeout = 50 ms
  Number of DMA buffers = 8
  Resulting cycle time = 50 ms
  Independently of the amount of data to be captured, the data callback will be called at least every 50 ms. Blocking the execution for more than 400 ms (8*50 ms) the DMA buffer resources will be exhausted.

  B) The configured DMA buffer size and the amount of data to be captured

  Example:
  Size of DMA buffer = 512 kByte
  Number of DMA buffers = 8
  Capture load = max. 50 MB/sec
  Resulting cycle time = 10 ms
  Due to the capture load every 10 ms one DMA buffer will be signaled. Blocking the execution for more than 80 ms (8*10 ms) the DMA buffer resources will be exhausted.

  Depending on the configuration and the amount of traffic to capture, blocking the execution for more than the resulting cycle time, may lead to exhaustion of DMA buffer resources. The resulting error will stop the running capture.

High load mode

Note: This function is available from Windows driver version 1.5.5. Use at least netANALYZER firmware version 1.5.4.

At very high data load occurring on all four ports simultaneously, the whole amount of data is too large be fully transmitted via PCI DMA. Therefore, the netANALYZER offers a special high load mode to be able to perform measurements under such conditions at all. However, this special high load mode reduces data load by truncating each frame length to 58 bytes. This mode is useful for generating frame statistics in cases where the frame payload is not important.

The status is always still valid (FCS, alignment errors …). Nevertheless, the following negative effects have to be taken into account:

- Truncated data is lost, also there is no possibility to get the original length of a specific frame.
- Wireshark would always indicate a frame size of 58 bytes and many frames would be incomplete. Therefore, using Wireshark in conjunction with the high load mode is not recommendable.

We recommend to use high load mode only in case all ports have a very high data load (near 100 %) at the same time.

3.3.20 netana_stop_capture

Stop a running live capture. This will also close the file in the filelist. To re-enable capturing to file after this call, you will need to call netana_set_filelist.

```
INT32 netana_stop_capture(NETANA_HANDLE hDev);
```

Parameter	Description
hDev	Device handle

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_INVALID_HANDLE (0x80200006)	Invalid handle
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_FUNCTION_FAILED (0x80200004)	Function failed Problem at reading or setting the status

3.3.21 netana_mngmt_exec_cmd

netana_mngmt_exex_cmd() provides a general function call interface. The command specified by ulCommandId specifies the function to execute.

```
INT32 netana_mngmt_exec_cmd (UINT32 ulCommandId,
                             void*  pvInputParameter,
                             UINT32 ulInputParameterSize,
                             void*  pvOutputParameter,
                             UIN32  ulOutputParameterSize)
```

Parameter	Description
ulCommandId	Number of the command (see table below)
pvInputParameter	depending on the command (see table below)
ulInputParameterSize	Size of related input parameter in units of bytes
pvOutputParameter	depending on the command (see table below)
ulOutputParameterSize	Size of related output parameter in units of bytes

The following table lists the available function commands provided by the general function call interface:

Command	Description
NETANA_MNGMT_CMD_BLINK	Blink command, see table Management Command NETANA_MNGMT_CMD_BLINK
NETANA_MNGMT_CMD_DEV_SCAN	Scan for new devices, see table Management Command NETANA_MNGMT_CMD_DEV_SCAN
NETANA_MNGMT_CMD_GET_DEV_FEATURE	Returns structure, containing information about the device features (see table Management Command NETANA_MNGMT_CMD_GET_DEV_FEATURE)
NETANA_MNGMT_CMD_SET_DEV_CLASS_FILTER	Allows altering the device filter of the driver (see table Management Command NETANA_MNGMT_CMD_SET_DEV_CLASS_FILTER).

Table 25: General Function Call Interface

NETANA_MNGMT_CMD_BLINK

Command ID		
NETANA_MNGMT_CMD_BLINK		
Function		
Execute the blink command on a specific device for device identification. Will blink the status LEDs on the device for 10 seconds		
Input parameters		
Element	**Type**	**Description**
szDevice	char*	Device name
Output parameters		
Element	**Type**	**Description**
none	none	none

Table 26: Management Command NETANA_MNGMT_CMD_BLINK

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter
NETANA_DEVICE_NOT_FOUND (0x80220003)	Device with the given name not found

NETANA_MNGMT_CMD_DEV_SCAN

Command ID		
NETANA_MNGMT_CMD_DEV_SCAN		
Function		
Scan for new devices.		
Input parameters		
Element	**Type**	**Description**
ptScanInParam	NETANA_MNGMT_DEV_SCAN_IN_T	Function pointer to call-back function for signaling bus scan state. For more information see NETANA_MNGMT_DEV_SCAN_IN_T on page 18.
Output parameters		
Element	**Type**	**Description**
ptScanOutParam	NETANA_MNGMT_DEV_SCAN_OUT_T	Number of found devices. For more information see NETANA_MNGMT_DEV_SCAN_OUT_T on page 18.

Table 27: Management Command NETANA_MNGMT_CMD_DEV_SCAN

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter
NETANA_DEVICE_NOT_FOUND (0x80220003)	Device with the given name not found

NETANA_MNGMT_CMD_GET_DEV_FEATURE

Command ID		
NETANA_MNGMT_CMD_GET_DEV_FEATURE		
Function		
Returns information structure of the device various features.		
Input parameters		
Element	**Type**	**Description**
ptFeatureIn	NETANA_MNGMT_GET_DEV_FEATURE_IN_T	Need to be set to the handle of the requested device. For more information see NETANA_MNGMT_GET_DEV_FEATURE_IN_T on page 18. (e.g. ptFeatureIn->hDev = hDeviceHandle)
Output parameters		
Element	**Type**	**Description**
ptFeatureOut	NETANA_MNGMT_GET_DEV_FEATURE_OUT_T	Strucutre contains information about the device features. For more information see NETANA_MNGMT_GET_DEV_FEATURE_OUT_T on page 19.

Table 28: Management Command NETANA_MNGMT_CMD_GET_DEV_FEATURE

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter
NETANA_DEVICE_NOT_FOUND (0x80220003)	Device with the given name not found

NETANA_MNGMT_CMD_SET_DEV_CLASS_FILTER

Command ID		
NETANA_MNGMT_CMD_SET_DEV_CLASS_FILTER		
Function		
Allows changing the device filter of the driver.		
Input parameters		
Element	**Type**	**Description**
ptFilterIn	NETANA_MNGMT_SET_DEV_CLASS_FILTER_IN_T	Mask need to be set to the device class to be filtered. For for more information see NETANA_MNGMT_SET_DEV_CLASS_FILTER_IN_T on page 20. (e.g. ptFilterIn->ulDeviceClass = NETANA_DEV_CLASS_NANL_500) (By default the driver filters for netANALYZER)
Output parameters		
Element	**Type**	**Description**
- / -	- / -	- / -

Table 29: Management Command NETANA_MNGMT_CMD_SET_DEV_CLASS_FILTER

Returns:

NETANA_NO_ERROR (0x0)	Success
NETANA_IOCTL_FAILED (0x80220001)	General error at sending of IOCTL
NETANA_INVALID_PARAMETER (0x80200001)	Invalid parameter
NETANA_DEVICE_NOT_FOUND (0x80220003)	Device with the given name not found

4 Example Code

The following examples explain the use of the netANALYZER API. They commonly require the following set of include statements:

```
#include "Windows.h"
#include "netana_user.h"
#include "netana_errors.h"

#include <stdio.h>
#include <conio.h>
#include <string>
#include <time.h>
```

4.1 Function `ConfigureGPIOs` - Configuring GPIO

The following piece of C code (function `ConfigureGPIOs`) can be used to configure the GPIOs on a netANALYZER card. It adjusts GPIOs 0 and 1.

```
void ConfigureGPIOs(NETANA_HANDLE hDevice)
{
  int32_t lResult;

  NETANA_GPIO_MODE_T tGpio = {0};

  /* Setup GPIO 0 to be captured on rising edge */
  tGpio.ulCaptureTriggers = NETANA_GPIO_TRIGGER_NONE;
  tGpio.ulMode            = NETANA_GPIO_MODE_RISING_EDGE;

  if(NETANA_NO_ERROR != (lResult = netana_set_gpio_mode(hDevice, 0, &tGpio)))
  {
    printf("Error configuring GPIO 0. ErrorCode=0x%08X\r\n", lResult);
  } else
  {
    printf("Configured GPIO 0 for rising edge detection\r\n");
  }

  /* Setup GPIO 1 to stop capturing (after 2 µs) if falling edge occurs */
  tGpio.ulCaptureTriggers = NETANA_GPIO_TRIGGER_STOP;
  tGpio.ulMode            = NETANA_GPIO_MODE_FALLING_EDGE;
  tGpio.ulEndDelay        = 200;

  if(NETANA_NO_ERROR != (lResult = netana_set_gpio_mode(hDevice, 1, &tGpio)))
  {
    printf("Error configuring GPIO 1. ErrorCode=0x%08X\r\n", lResult);
  } else
  {
    printf("Configured GPIO 1 to stop capturing after 2 µs when a falling edge is
detected\r\n");
  }
}
```

4.2 Function `ConfigureFilters` - Setting Filters

The following piece of C code (function `ConfigureFilters`) can be used to configure filters on a netANALYZER card. It configures the port 0 of a card in such manner, that only TCP/IP and ARP telegrams are recorded.

```
void ConfigureFilters(NETANA_HANDLE hDevice)
{
  NETANA_FILTER_T tFilterA = {0};
  uint8_t         abFilterMaskA[16]  = {0};
  uint8_t         abFilterValueA[16] = {0};

  NETANA_FILTER_T tFilterB = {0};
  uint8_t         abFilterMaskB[16]  = {0};
  uint8_t         abFilterValueB[16] = {0};

  uint32_t        ulRelation;

  int32_t         lResult;

  /* Filter A matches TCP/IP Frames (Frametype = 0x0800) */
  abFilterMaskA[12]  = 0xFF;
  abFilterMaskA[13]  = 0xFF;
  abFilterValueA[12] = 0x08;
  abFilterValueA[13] = 0x00;
  tFilterA.ulFilterSize = sizeof(abFilterMaskA);
  tFilterA.pbMask  = abFilterMaskA;
  tFilterA.pbValue = abFilterValueA;

  /* Filter B matches ARP Frames (Frametype = 0x0806) */
  abFilterMaskB[12]  = 0xFF;
  abFilterMaskB[13]  = 0xFF;
  abFilterValueB[12] = 0x08;
  abFilterValueB[13] = 0x06;
  tFilterB.ulFilterSize = sizeof(abFilterMaskB);
  tFilterB.pbMask  = abFilterMaskB;
  tFilterB.pbValue = abFilterValueB;

  ulRelation =  NETANA_FILTER_RELATION_FILTER_A_ENABLE |
                NETANA_FILTER_RELATION_FILTER_B_ENABLE |
                NETANA_FILTER_RELATION_A_OR_B          |
                NETANA_FILTER_RELATION_ACCEPT_FILTER;

  /* Setup filters on Port 0 to only capture TCP/IP Frames
    (Frametype = 0x0800) and ARP (Type 0x0806) frames */
  if(NETANA_NO_ERROR != (lResult = netana_set_filter(hDevice,
                                                     0,
                                                     &tFilterA,
                                                     &tFilterB,
                                                     ulRelation)))
  {
    printf("Error setting filters on port 0. ErrorCode=0x%08X\r\n", lResult);

  } else
  {
    printf("Successfully set filters on Port 0\r\n");
  }
}
```

4.3 Function DoCapture - Capturing

This piece of C code (function DoCapture) can be used to start capturing data. It starts the capturing process with data transfer by callback (via DataCallback()) and status callback (via StatusCallback()).

Note: If you want to capture to a file, you will need to call netana_set_filelist before starting the capture e.g.

```
netana_set_filelist(hDevice, "C:\\","Capture", 1, 1 * 1024 * 1024
* 1024);
```

```
void DoCapture(NETANA_HANDLE hDevice)
{
  int32_t  lResult;
  /* Reference time for wireshark needs to be UNIX Timestamp (seconds sind 1.1.1970)
     and as we are using a nanosecond timestamp, we need to multiply it with 1000000000
*/
  uint64_t ullReferenceTime = _time64(NULL) * 1000 * 1000 * 1000;

  /* NOTE: If you want to capture to a file, you will need to call netana_set_filelist
before
           starting the capture.
           e.g.
           netana_set_filelist(hDevice, "C:\\", "Capture", 1, 1 * 1024 * 1024 * 1024);
  */
  if(NETANA_NO_ERROR != (lResult = netana_start_capture(hDevice,
                                                        0,
                                                        0xF,
                                                        NETANA_MACMODE_ETHERNET,
                                                        ullReferenceTime,
                                                        StatusCallback,
                                                        DataCallback,
                                                        NULL)))
  {
    printf("Error starting capture. ErrorCode=0x%08X\r\n", lResult);

  } else
  {
    printf("!!!Press any key to stop capturing!!!\r\n");

    while(!_kbhit())
    {
      if(s_fCaptureStopPending)
      {
        printf("netANALYZER Firmware detected an error and stopped capturing!\r\n");
        printf("Aborting capture!\r\n");
        break;
      }

      Sleep(10);
    }

    /* NOTE: We need to call netana_stop_capture even if the firmware stopped
automatically, to tell the firmware we've understood that capturing was automatically
stopped */
    netana_stop_capture(hDevice);
    printf("Stopped Capturing!\r\n");
  }
}
static bool s_fCaptureStopPending = false;

//=============================================================================
// Status change callback. Called by the driver if a function pointer was
// passed in netana_start_capture and the state of the card has changed
//
//=============================================================================
```

```
static void APIENTRY StatusCallback(uint32_t ulCaptureState, uint32_t ulCaptureError,
void* /*pvUser*/)
{
  switch(ulCaptureState)
  {
  case NETANA_CAPTURE_STATE_OFF:
    printf("Capture Stopped. ErrorCode=0x%08X\r\n", ulCaptureError);
    break;

  case NETANA_CAPTURE_STATE_START_PENDING:
    printf("Preparing Capture Start.\r\n");
    break;

  case NETANA_CAPTURE_STATE_RUNNING:
    printf("Capture Running.\r\n");
    break;

  case NETANA_CAPTURE_STATE_STOP_PENDING:
    printf("Capture Stop Pending. ErrorCode=0x%08X\r\n", ulCaptureError);
    s_fCaptureStopPending = true;
    break;

  default:
    printf("Unknown Capture State (%u). ErrorCode=0x%08X\r\n", ulCaptureState,
ulCaptureError);
    break;
  }
}

//============================================================================
// New data indication callback. Called by the driver when new capture data has //
arrived.
//
//============================================================================
static void APIENTRY DataCallback(void* pvBuffer, uint32_t ulDataSize, void* /*pvUser*/)
{
  uint8_t*  pbBuffer   = (uint8_t*)pvBuffer;
  uint32_t  ulOffset   = 0;
  uint32_t  ulFrameCnt = 0;

  while(ulOffset < ulDataSize)
  {
    NETANA_FRAME_HEADER_T* ptFrame  = (NETANA_FRAME_HEADER_T*)(pbBuffer + ulOffset);
    uint32_t ulFrameLen = (ptFrame->ulHeader & NETANA_FRAME_HEADER_LENGTH_MSK) >>
NETANA_FRAME_HEADER_LENGTH_SRT;

    ulFrameCnt++;

    /* Adjust Offset to next DWORD aligned address */
    ulOffset += sizeof(*ptFrame) + ulFrameLen;
    while(ulOffset % 4)
      ++ulOffset;
  }

  printf("Received %u frames. DataLen=%u\r\n", ulFrameCnt, ulDataSize); }
```

4.4 Main Program

This piece of C code can be used as main program to execute the functions

- ConfigureGPIOs()
- ConfigureFilters()
- DoCapture()

described above.

```
//=============================================================================
// Main
//
//
//=============================================================================
int _tmain(int /*argc*/, _TCHAR* /*argv[]*/)
{
  int32_t                     lResult;
  NETANA_DRIVER_INFORMATION_T tDriverInfo = {0};
  char*                       szDeviceToUse = NULL;

  /* Try to open the driver */
  if(NETANA_NO_ERROR != (lResult = netana_driver_information(sizeof(tDriverInfo),
                                                             &tDriverInfo)))
  {
    printf("Error opening driver. ErrorCode=0x%08X\r\n", lResult);
  } else
  {
    printf("Driver Information:\r\n");
    printf("-------------------\r\n");
    printf(" Version     : %u.%u.%u.%u\r\n", tDriverInfo.ulVersionMajor,
                                             tDriverInfo.ulVersionMinor,
                                             tDriverInfo.ulVersionBuild,
                                             tDriverInfo.ulVersionRevision);

    printf(" Cards       : %u\r\n", tDriverInfo.ulCardCnt);
    printf(" DMA Buffers : %u x %u Bytes\r\n", tDriverInfo.ulDMABufferCount,
tDriverInfo.ulDMABufferSize);
    printf(" Max Files   : %u\r\n", tDriverInfo.ulMaxFileCount);

    printf(" Found cards : \r\n");

    /* Enumerate all available boards and use the first found one, to do our tests */
    for(uint32_t ulCard = 0; ulCard < tDriverInfo.ulCardCnt; ulCard++)
    {
      NETANA_DEVICE_INFORMATION_T tDevInfo = {0};

      if(NETANA_NO_ERROR != (lResult = netana_enum_device(ulCard, sizeof(tDevInfo),
&tDevInfo)))
      {
        printf("[%u]: Error enumerating card #%u. ErrorCode=0x%08X\r\n",
               ulCard, ulCard, lResult);

      } else
      {
        if(NULL == szDeviceToUse)
        {
          /* NOTE: We will always use the first available device for our tests */
          szDeviceToUse = _strdup((char*)tDevInfo.szDeviceName);
        }

        printf("[%u]: DeviceName = '%s'\r\n", ulCard, (char*)tDevInfo.szDeviceName);
        printf("      DeviceNr = %u, SerialNr = %u\r\n",
               tDevInfo.ulDeviceNr,
               tDevInfo.ulSerialNr);
        printf("      Firmware = %s V%u.%u.%u.%u\r\n",
```

```
               (char*)tDevInfo.szFirmwareName,
               tDevInfo.ulVersionMajor,
               tDevInfo.ulVersionMinor,
               tDevInfo.ulVersionBuild,
               tDevInfo.ulVersionRevision);
        printf("      Ports = %u, GPIOs = %u, FilterSize = %u\r\n",
               tDevInfo.ulPortCnt,
               tDevInfo.ulGpioCnt,
               tDevInfo.ulFilterSize);
      }
    }

    if(NULL == szDeviceToUse)
    {
      printf("No device found for further testing\r\n");

    } else
    {
      NETANA_HANDLE hDevice = NULL;

      /* Get a handle to the device */
      if(NETANA_NO_ERROR != (lResult = netana_open_device(szDeviceToUse, &hDevice)))
      {
        printf("Error opening device '%s'. ErrorCode=0x%08X\r\n", szDeviceToUse,
lResult);

      } else
      {
        printf("Starting tests on Device '%s'\r\n", szDeviceToUse);
        printf("----------------------------------------\r\n");

        ConfigureGPIOs(hDevice);
        ConfigureFilters(hDevice);

        DoCapture(hDevice);

        printf("----------------------------------------\r\n");
        printf("Test ended!\r\n");
        netana_close_device(hDevice);
      }
    }
  }

     return 0;
}
```

5 Error List

Generally, there are two separate kinds of errors within the netANALYZER API:

- netANALYZER Device Driver Errors (on page 54)
- Capturing Errors (on page 58)

5.1 netANALYZER Device Driver Errors

The netANALYZER Device Driver Error Numbers are built according to the following table:

Bit	Description
D15 - D0	Code This item represents facility's individual status code
D27 – D16	Facility This item represents the facility code according to *Table 32: Classification of facility codes*.
D28	Reserved bit This item contains a reserved bit
D29	Customer code flag
D31 – D30	Severity code The severity code classifies the error messages according to their severity according to *Table 31: Classification of severity codes*.

Table 30: Systematics of netANALYZER Device Driver Error Numbers

The following table explains the possible severity codes:

D31	D30	Error classification
0	0	Success
0	1	Informational
1	0	Warning
1	1	Error

Table 31: Classification of severity codes

The following table explains the possible facility codes:

Facility code	Source facility of error
0	`FACILITY_NULL`
32	`NETANA_GENERIC`
33	`NETANA_TOOLKIT`
34	`NETANA_DRIVER`

Table 32: Classification of facility codes

5.1.1 Generic Errors

These errors are associated with facility code NETANA_GENERIC = 32.

Value	Definition	Description
0x00000000	NETANA_NO_ERROR	No Error
0x80200001	NETANA_INVALID_PARAMETER	Invalid parameter
0x80200002	NETANA_INVALID_PORT	Invalid port number
0x80200003	NETANA_OUT_OF_MEMORY	Out of memory
0x80200004	NETANA_FUNCTION_FAILED	Function failed (Generic Error)
0x80200005	NETANA_INVALID_POINTER	Invalid pointer
0x80200006	NETANA_INVALID_HANDLE	Invalid handle
0x80200007	NETANA_NO_WORKING_DIRECTORY	No working directory
0x80200008	NETANA_NO_ENTRY_FOUND	No entry found
0x80200009	NETANA_FUNCTION_NOT_AVAILABLE	Function not available

Table 33: List of Generic Errors

5.1.2 Toolkit Errors

These errors are associated with facility code NETANA_TOOLKIT = 33.

Value	Definition	Description
0x80210001	NETANA_TKIT_INITIALIZATION_FAILED	Toolkit initialization failed
0x80210002	NETANA_DMABUFFER_CREATION_FAILED	Creation of DMA buffers failed
0x80210003	NETANA_HWRESET_ERROR	Error during hardware reset of device
0x80210004	NETANA_CHIP_NOT_SUPPORTED	Chip type is not supported by toolkit
0x80210005	NETANA_DOWNLOAD_FAILED	Download of Bootloader / Firmware failed
0x80210006	NETANA_FW_START_FAILED	Error starting firmware
0x80210007	NETANA_DEV_MAILBOX_FULL	Device mailbox is full
0x80210008	NETANA_DEV_NOT_READY	Device not ready
0x80210009	NETANA_DEV_MAILBOX_TOO_SHORT	Mailbox is too short for packet
0x8021000A	NETANA_DEV_GET_NO_PACKET	No packet available
0x8021000B	NETANA_BUFFER_TOO_SHORT	Given buffer is too short
0x8021000C	NETANA_TRANSFER_TIMEOUT	Transfer timed out
0x8021000D	NETANA_IRQEVENT_CREATION_FAILED	Error creating interrupt events
0x8021000E	NETANA_IRQLOCK_CREATION_FAILED	Error creating internal IRQ locks

Table 34: List of Toolkit Errors

5.1.3 Driver Errors

These errors are associated with facility code NETANA_DRIVER = 34.

Value	Definition	Description
0x80220001	NETANA_IOCTL_FAILED	Error sending IOCTL to driver
0x80220002	NETANA_DRIVER_NOT_RUNNING	netANALYZER Driver is not running
0x80220003	NETANA_DEVICE_NOT_FOUND	Device with the given name does not exist
0x80220004	NETANA_DEVICE_STILL_OPEN	Device is still in use by another application
0x80220005	NETANA_DEVICE_NOT_OPEN	Device is not open
0x80220006	NETANA_MEMORY_MAPPING_FAILED	Failed to map memory
0x80220007	NETANA_FILE_OPEN_ERROR	Error opening file
0x80220008	NETANA_FILE_READ_FAILED	Error reading file
0x80220009	NETANA_FILE_CREATION_FAILED	Error creating file
0x8022000A	NETANA_FILE_WRITE_FAILED	Error writing file
0x8022000B	NETANA_CAPTURE_ACTIVE	Capturing is currently active
0x8022000C	NETANA_CAPTURE_NOT_ACTIVE	Capturing is currently stopped
0x8022000D	NETANA_FILECAPTURE_NOT_ACTIVE	Capturing to file is not enabled
0x8022000E	NETANA_CONFIGURATION_ERROR	Capture configuration error
0x8022000F	NETANA_THREAD_CREATION_FAILED	Creation of worker thread failed
0x80220010	NETANA_NO_BUFFER_DATA	Buffer content not valid
0x80220011	NETANA_NO_STATE_CHANGE	No state change
0x80220012	NETANA_NO_PLUGIN_FOUND	No plug-in found

Table 35 : List of Driver Errors

5.1.4 Transport Errors

These errors are associated with facility code NETANA_DRIVER = 35.

Value	Definition	Description
0x80230001	NETANA_TRANSPORT_CHECKSUM_ERROR	Checksum incorrect
0x80230002	NETANA_TRANSPORT_LENGTH_INCOMPLETE	Transport length is incomplete
0x80230003	NETANA_TRANSPORT_DATA_TYPE_UNKOWN	Unknown data type
0x80230004	NETANA_TRANSPORT_DEVICE_UNKNOWN	Device is unknown
0x80230005	NETANA_TRANSPORT_CHANNEL_UNKNOWN	Channel is unknown
0x80230006	NETANA_TRANSPORT_SEQUENCE	Sequence error
0x80230007	NETANA_TRANSPORT_BUFFEROVERFLOW	Buffer overflow
0x80230008	NETANA_TRANSPORT_KEEPALIVE	Keep alive error
0x80230009	NETANA_TRANSPORT_RESOURCE	Resource error
0x8023000A	NETANA_TRANSPORT_ERROR_UNKNOWN	Unknown error
0x8023000B	NETANA_TRANSPORT_RECV_TIMEOUT	Timeout while receiving data
0x8023000C	NETANA_TRANSPORT_SEND_TIMEOUT	Timeout while sending data
0x8023000D	NETANA_TRANSPORT_CONNECT	Unable to communicate to the device / no answer
0x8023000E	NETANA_TRANSPORT_ABORTED	Transfer has been aborted due to keep alive timeout or interface detachment"
0x8023000F	NETANA_TRANSPORT_INVALID_RESPONSE	The packet response was rejected due to invalid packet data
0x80230010	NETANA_TRANSPORT_UNKNOWN_DATALAYER	The data layer provided by the device is not supported
0x80230011	NETANA_CONNECTOR_FUNCTIONS_READ_ERROR	Error reading the connector functions from the DLL
0x80230012	NETANA_CONNECTOR_IDENTIFIER_TOO_LONG	Connector delivers an identifier longer than 6 characters
0x80230013	NETANA_CONNECTOR_IDENTIFIER_EMPTY	Connector delivers an empty identifier
0x80230014	NETANA_CONNECTOR_DUPLICATE_IDENTIFIER	Connector identifier already used

Table 36: List of Transport Errors

5.1.5 Transport Header State Errors

These errors are associated with facility code NETANA_DRIVER = 35.

Value	Definition	Description
0x80230024	NETANA_TRANSPORT_DATA_TOO_SHORT	Received transaction data too short
0x80230025	NETANA_TRANSPORT_UNSUPPORTED_FUNCTION	Function is not supported
0x80230026	NETANA_TRANSPORT_TIMEOUT	Transport timeout

Table 37: List of Transport Header State Errors

5.1.6 Marshaller Target Error

This error is associated with facility code NETANA_DRIVER = 35.

Value	Definition	Description
0xC0230001	NETANA_CAPTURE_ERROR_ON_TARGET	Capture error on the target device

Table 38: List of Marshaller Target Errors

5.2 Capturing Errors

These are low-level errors detected by netANALYZER hardware.

Capture Errors		
0x00000000	NETANA_CAPTURE_ERROR_STOP_TRIGGER	No error. (Capturing stopped by user or GPIO trigger)
0xC0660004	NETANA_CAPTURE_ERROR_NO_DMACHANNEL	No free DMA channel available. Probably host is too slow
0xC0660005	NETANA_CAPTURE_ERROR_URX_OVERFLOW	XC buffer overflow (URX overflow) Occurs because a non IEEE802.3 conform traffic is captured (e.g. too short frames, too small IFG).
0xC066000B	NETANA_CAPTURE_ERROR_NO_HOSTBUFFER	No free DMA buffer available. Host is too slow to handle data efficiently.
0xC066000C	NETANA_CAPTURE_ERROR_NO_INTRAMBUFFER	Internal capture buffer overflow (no free INTRAM). Firmware is out of memory resources and is unable to buffer more data. This may also be caused by a slow file system or a slow application
0xC066000D	NETANA_CAPTURE_ERROR_FIFO_FULL	Firmware is out of FIFO resources and is unable to buffer more data. This may also be caused by a slow file system or a slow application
0xC0770000	NETANA_CAPTURE_ERROR_DRIVER_FILE_FULL	End of capture file reached. Driver has stopped capturing
0xC0770001	NETANA_CAPTURE_ERROR_DRIVER_INVALID_BUFFERSIZE	Internal capture buffer overflow (no free INTRAM)

Table 39 : List of Capturing Errors

6 Appendix

6.1 List of Tables

6.2 List of Figures

6.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
New Delhi - 110 065
Phone: +91 11 26915430
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com