



Driver Manual
netANALYZER API
Windows 2000/XP/Vista/7
V1.3

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC091113DRV03EN | Revision 3 | English | 2010-02 | Released | Public

Table of Contents

1	Introduction.....	3
1.1	About this Document.....	3
1.2	List of Revisions.....	3
1.3	Terms, Abbreviations and Definitions.....	3
1.4	Legal Notes.....	4
1.4.1	Copyright.....	4
1.4.2	Important Notes.....	4
1.4.3	Exclusion of Liability.....	5
1.4.4	Export.....	5
2	Overview.....	6
2.1	Ring Buffer Mode.....	7
2.2	Capture File Structure.....	8
3	API Definitions.....	10
3.1	Structures.....	10
3.1.1	NETANA_FRAME_HEADER_T.....	10
3.1.2	NETANA_DRIVER_INFORMATION_T.....	11
3.1.3	NETANA_DEVICE_INFORMATION_T.....	12
3.1.4	NETANA_FILTER_T.....	13
3.1.5	NETANA_GPIO_MODE_T.....	13
3.1.6	NETANA_PORT_STATE_T.....	14
3.1.7	NETANA_FILE_INFORMATION_T.....	14
3.2	Callbacks.....	15
3.2.1	PFN_STATUS_CALLBACK.....	15
3.2.2	PFN_DATA_CALLBACK.....	16
3.3	Functions.....	17
3.3.1	netana_driver_information.....	18
3.3.2	netana_get_error_description.....	18
3.3.3	netana_enum_device.....	19
3.3.4	netana_open_device.....	20
3.3.5	netana_close_device.....	21
3.3.6	netana_device_info.....	21
3.3.7	netana_get_portstat.....	22
3.3.8	netana_get_state.....	23
3.3.9	netana_access_phy_reg.....	24
3.3.10	netana_set_filelist.....	25
3.3.11	netana_get_gpio_mode.....	26
3.3.12	netana_set_gpio_mode.....	26
3.3.13	netana_get_filter.....	27
3.3.14	netana_set_filter.....	28
3.3.15	netana_file_info.....	29
3.3.16	netana_start_capture.....	30
3.3.17	netana_stop_capture.....	31
4	Example Code.....	32
4.1	Function ConfigureGPIOs - Configuring GPIO.....	32
4.2	Function ConfigureFilters - Setting Filters.....	33
4.3	Function DoCapture - Capturing.....	34
4.4	Main Program.....	36
5	Error List.....	38
5.1	netANALYZER Device Driver Errors.....	38
5.1.1	Generic Errors.....	39
5.1.2	Toolkit Errors.....	39
5.1.3	Driver Errors.....	40
5.2	Capturing Errors.....	41
6	Appendix.....	42
6.1	List of Tables.....	42
6.2	List of Figures.....	42
6.3	Contacts.....	43

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

Table 1: List of Revisions

1.3 Terms, Abbreviations and Definitions

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

Table 2: Terms, Abbreviations and Definitions

1.4 Legal Notes

1.4.1 Copyright

© 2008-2010 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying material (user manual, accompanying texts, documentation, etc.) are protected by German and international copyright law as well as international trade and protection provisions. You are not authorized to duplicate these in whole or in part using technical or mechanical methods (printing, photocopying or other methods), to manipulate or transfer using electronic systems without prior written consent. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. The included diagrams do not take the patent situation into account. The company names and product descriptions included in this document may be trademarks or brands of the respective owners and may be trademarked or patented. Any form of further use requires the explicit consent of the respective rights owner.

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 countries, 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:

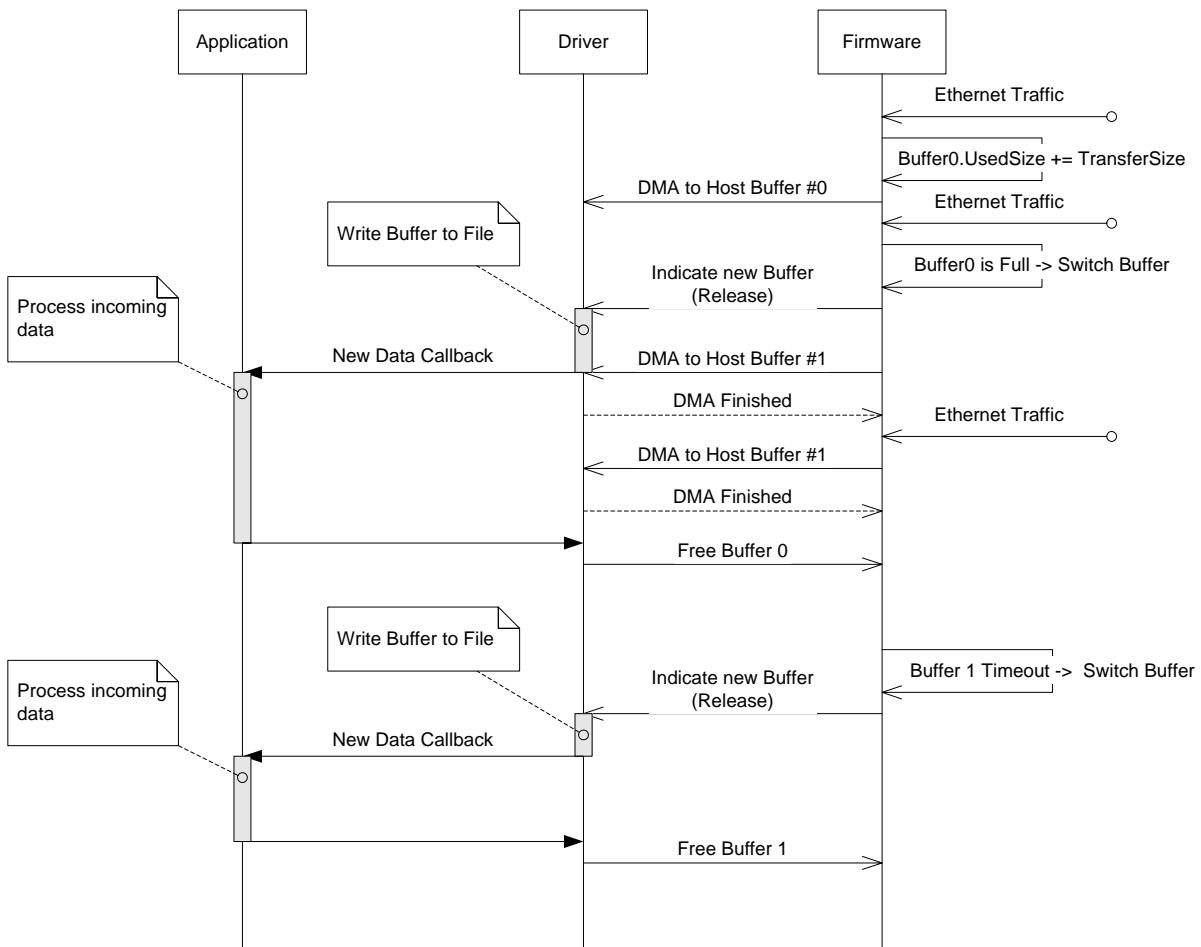


Figure 1: Overview - Capturing Process

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:

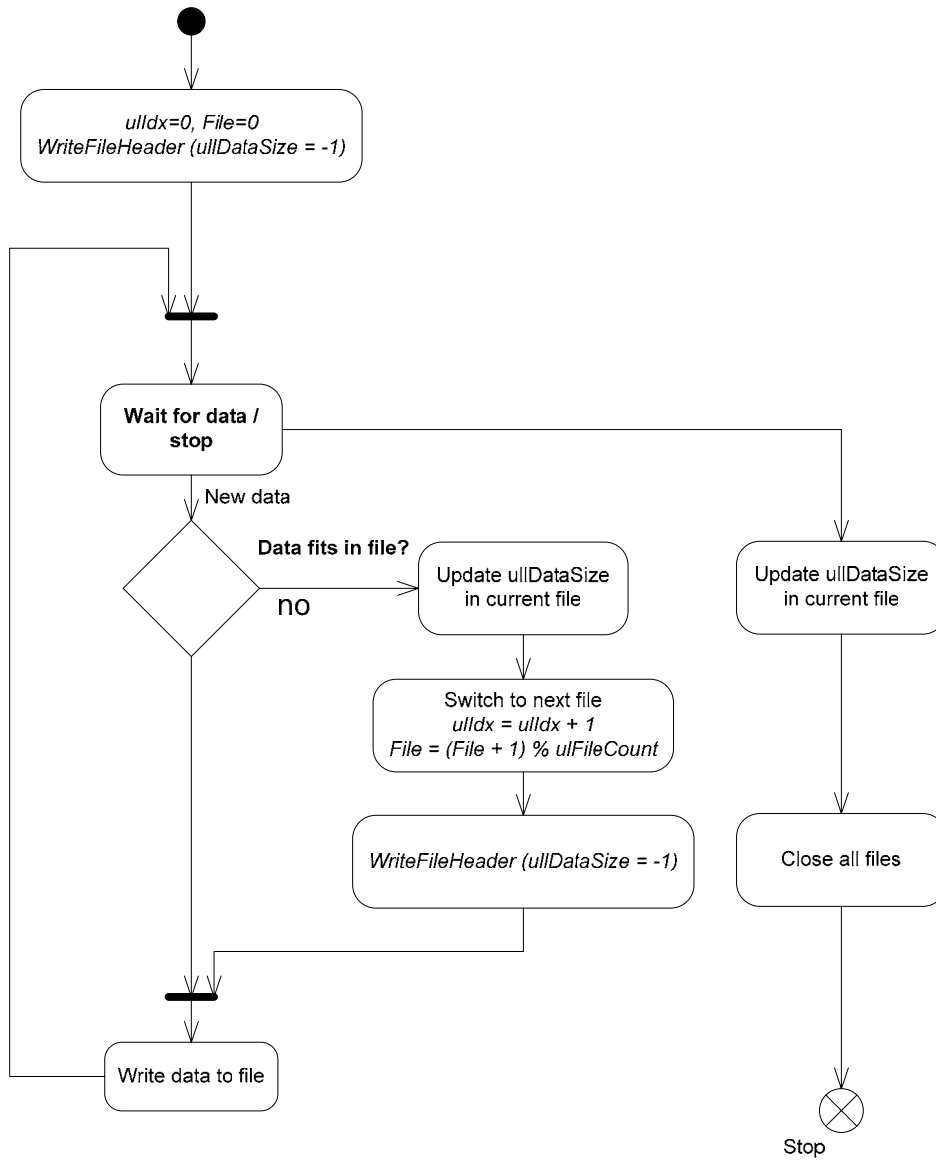


Figure 2: Ring Buffer Mode State Diagram

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 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.

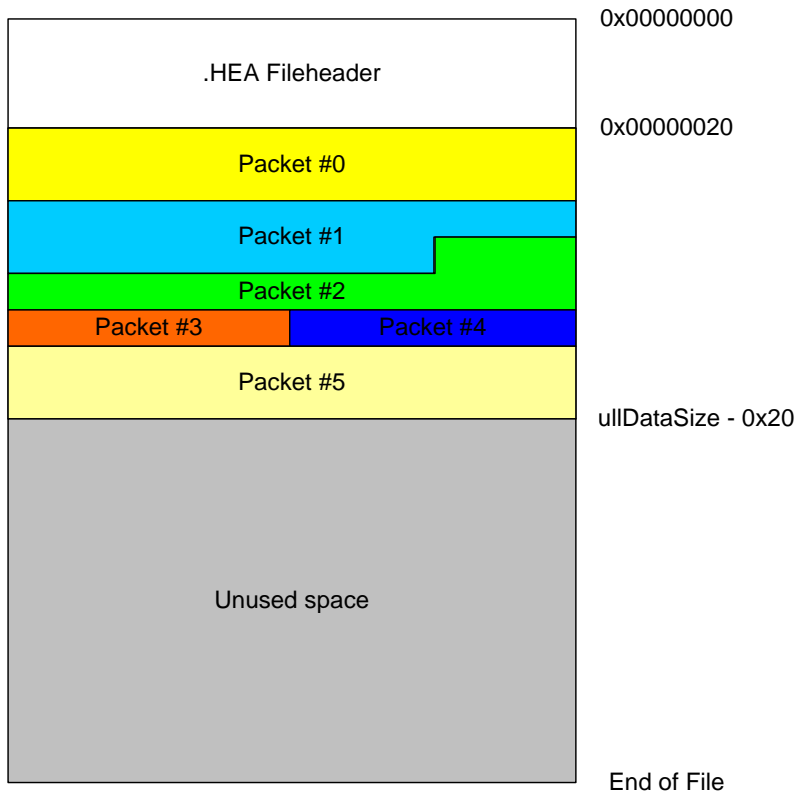


Figure 3: Capture File Overview - V1.3 and later

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 (nano-seconds 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	Timezone 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 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 ullIdx 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 <i>Table 6: Frame Error Codes</i> below Bit 8 : 0=Ethernet Port, 1=GPIO Port Bit 9 : 0 = Ethernet mode, 1=Transparent mode 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 <code>netana_start_capture</code>
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_LO NG_PRE AMBLE	MSK_SHO RT_PREA MBLE	MSK_SHO RT_FRAM E	MSK_SF D_ERRO R	MSK_TO O_LONG	MSK_FCS _ERROR	MSK_ALI GN_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_driver_information" (section NETANA_DEVICE_INFORMATION_T on page 12) 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

Table 7: Structure: Driver Information

3.1.3 NETANA_DEVICE_INFORMATION_T

Defines all features of a netANALYZER device and is used in the following functions:

- `netana_enum_device` (see section *netana_enum_device* on page 19)
- `netana_device_info` (see section *netana_device_info* on page 21)

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

Table 8: Structure: Device Information

3.1.4 NETANA_FILTER_T

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 27)
- `netana_set_filter` (see section *netana_set_filter* on page 28)

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

Table 9: Structure: Filter Description

3.1.5 NETANA_GPIO_MODE_T

Defines the capturing / trigger of a GPIO and is used in the following functions:

- `netana_get_gpio_mode` (see section *netana_get_gpio_mode* on page 26)
- `netana_set_gpio_mode` (see section *netana_set_gpio_mode* on page 26)

Element	Type	Description
<code>ulMode</code>	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
<code>ulCaptureTriggers</code>	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
<code>ulEndDelay</code>	UINT32	Delay in multiple of 10 ns after capturing is stopped. Note: This value will be used if NETANA_GPIO_TRIGGER_STOP is set for this GPIO

Table 10: Structure: GPIO Description

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 22) function.

Element	Type	Description
ullLinkState	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)
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 11: Structure: Portstate Description

3.1.7 NETANA_FILE_INFORMATION_T

File capture statistics for currently running capture used in the "netana_file_info" (see section *netana_file_info* on page 29) 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 12: Structure: File Information 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.

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 too 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.
pvUser	User specific parameter passed to netana_start_capture

Table 13: Status Callback - Parameters

3.2.2 PFN_DATA_CALLBACK

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

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 <i>NETANA_FRAME_HEADER_T</i> on page 10.
ulDataLen	Length of data in buffer
pvUser	User specific parameter passed to <i>netana_start_capture</i>

Table 14: Data 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 (Firmwareversion, 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_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

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	Success
NETANA_DRIVER_NOT_RUNNING	Driver is not running

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	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	Success
NETANA_DEVICE_NOT_FOUND	Device with card number was not found

Example:

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

while(NETANA_NO_ERROR == 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	Success
NETANA_DEVICE_NOT_FOUND	Device with given name was not found

Example:

```
NETANA_HANDLE hDev;  
  
if (NETANA_NO_ERROR == 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	Success

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	Success

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	Success

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	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	Success

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.he

C:\capture_1.he

```
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	Success

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	Success

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
ptMode	GPIO configuration

Returns:

NETANA_NO_ERROR	Success

3.3.13 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 = FilterA AND Filter B 1 = Result = FilterA OR Filter B 0x00000080 NETANA_FILTER_RELATION_REJECT_FILTER 0 = Accept frames if filter matches 1 = Reject frames if filter matches 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 B 1 = Negate Filter B 0x80000000 NETANA_FILTER_RELATION_FILTER_A_ENABLE 0 = Do not enable Filter B 1 = Enable Filter B

Returns:

NETANA_NO_ERROR	Success

3.3.14 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 (Bitmask) 0x00000001 NETANA_FILTER_RELATION_A_OR_B 0 = Result = FilterA AND Filter B 1 = Result = FilterA OR Filter B 0x00000080 NETANA_FILTER_RELATION_REJECT_FILTER 0 = Accept frames if filter matches 1 = Reject frames if filter matches 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 B 1 = Negate Filter B 0x80000000 NETANA_FILTER_RELATION_FILTER_A_ENABLE 0 = Do not enable Filter B 1 = Enable Filter B

Returns:

NETANA_NO_ERROR	Success

3.3.15 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	Success

3.3.16 netana_start_capture

Start a live capture. This function must be called after all ports and GPIOs have been correctly configured. If capturing the file is needed, the function `netana_set_filelist` must be called before every call to this function.

```
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. 0x80000000: NETANA_CAPTUREMODE_RINGBUFFER - Set this bit to enable data capturing in ring buffermode (only valid for capturing to file)
ulPortEnableMask	Bitmask of enabled Ethernet ports. Each bit represents a port.
ulMacMode	0: NETANA_MACMODE_ETHERNET - Capture in Ethernet mode (802.3 frames) 1: NETANA_MACMODE_TRANSPARENT - Capture all bytes on wire without regarding any specifications
ullReferenceTime	Capture reference time in nanoseconds. This is expected to be the UNIX time (Seconds since 1.1.1970) in nanoseconds, as Wireshark expects it in 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 be set to any customer value.
pfnStatus	Status change notification callback. Set to NULL if not needed.
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 <code>netana_set_filelist</code> .
pvUser	User parameter passed on callbacks

Returns:

NETANA_NO_ERROR	Success

3.3.17 netana_stop_capture

Stop a running live capture. This will also close and 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	Success

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 since 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 38)
- Capturing Errors (on page 41)

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 <i>Table 17: Classification of facility codes.</i>
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 <i>Table 16: Classification of severity codes.</i>

Table 15: 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 16: 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 17: Classification of facility codes

5.1.1 Generic Errors

These errors are associated with facility code `NETANA_GENERIC` = 32.

Value	Definition	Description
0x00000000	<code>NETANA_NO_ERROR</code>	No Error
0x80200001	<code>NETANA_INVALID_PARAMETER</code>	Invalid parameter
0x80200002	<code>NETANA_INVALID_PORT</code>	Invalid port number given
0x80200003	<code>NETANA_OUT_OF_MEMORY</code>	Out of memory
0x80200004	<code>NETANA_FUNCTION_FAILED</code>	Function failed
0x80200005	<code>NETANA_INVALID_POINTER</code>	Invalid pointer

Table 18: List of Generic Errors

5.1.2 Toolkit Errors

These errors are associated with facility code `NETANA_TOOLKIT` = 33.

Value	Definition	Description
0x80210001	<code>NETANA_TKIT_INITIALIZATION_FAILED</code>	Toolkit initialization failed
0x80210002	<code>NETANA_DMABUFFER_CREATION_FAILED</code>	Error creating DMA buffers
0x80210003	<code>NETANA_HWRESET_ERROR</code>	netX hardware reset failed
0x80210004	<code>NETANA_CHIP_NOT_SUPPORTED</code>	The detected netX chip is not supported by toolkit
0x80210005	<code>NETANA_DOWNLOAD_FAILED</code>	Error downloading firmware
0x80210006	<code>NETANA_FW_START_FAILED</code>	Error starting firmware
0x80210007	<code>NETANA_DEV_MAILBOX_FULL</code>	The device mailbox is full
0x80210008	<code>NETANA_DEV_NOT_READY</code>	The device is not ready
0x80210009	<code>NETANA_DEV_MAILBOX_TOO_SHORT</code>	The device mailbox is too small for receiving the entire packet
0x8021000A	<code>NETANA_DEV_GET_NO_PACKET</code>	There is no packet available on device
0x8021000B	<code>NETANA_BUFFER_TOO_SHORT</code>	The given buffer is too short for receiving the entire packet
0x8021000C	<code>NETANA_TRANSFER_TIMEOUT</code>	A time out occurred during packet transfer
0x8021000D	<code>NETANA_IRQEVENT_CREATION_FAILED</code>	An error occurred creating interrupt events
0x8021000E	<code>NETANA_IRQLOCK_CREATION_FAILED</code>	An error occurred creating internal IRQ locks

Table 19: 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	An error occurred sending IOCTL to driver
0x80220002	NETANA_DRIVER_NOT_RUNNING	The netANALYZER Device Driver is not running
0x80220003	NETANA_DEVICE_NOT_FOUND	A device with the specified name does not exist
0x80220004	NETANA_DEVICE_STILL_OPEN	The device is still in use by another process
0x80220005	NETANA_DEVICE_NOT_OPEN	The device has not been opened
0x80220006	NETANA_MEMORY_MAPPING_FAILED	An error occurred mapping memory to the user application
0x80220007	NETANA_FILE_OPEN_ERROR	An error occurred opening a capturing file
0x80220008	NETANA_FILE_READ_FAILED	An error occurred reading a capturing file
0x80220009	NETANA_FILE_CREATION_FAILED	The creation of the capturing file failed.
0x8022000A	NETANA_FILE_WRITE_FAILED	An error occurred during writing to the capturing 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	There is an error in the capturing configuration
0x8022000F	NETANA_THREAD_CREATION_FAILED	An error occurred during creating a worker thread
0x80220010	NETANA_NO_BUFFER_DATA	There are currently no new data available from DMA buffer
0x80220011	NETANA_NO_STATE_CHANGE	There are currently no state change data available

Table 20 : List of Driver Errors

5.2 Capturing Errors

These are low-level errors detected by netANALYZER hardware.

Capture Errors		
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

Table 21 : List of Capturing Errors

6 Appendix

6.1 List of Tables

Table 1: List of Revisions	3
Table 2: Terms, Abbreviations and Definitions.....	3
Table 3: Structure of .HEA Header.....	8
Table 4: Structure Capture File Data (V1.3 and later)	9
Table 5: Structure: Capture Data Header.....	10
Table 6: Frame Error Codes.....	10
Table 7: Structure: Driver Information	11
Table 8: Structure: Device Information.....	12
Table 9: Structure: Filter Description.....	13
Table 10: Structure: GPIO Description.....	13
Table 11: Structure: Portstate Description.....	14
Table 12: Structure: File Information Description	14
Table 13: Status Callback - Parameters.....	15
Table 14: Data Callback - Parameters	16
Table 15: Systematics of netANALYZER Device Driver Error Numbers	38
Table 16: Classification of severity codes	38
Table 17: Classification of facility codes.....	38
Table 18: List of Generic Errors	39
Table 19: List of Toolkit Errors	39
Table 20 : List of Driver Errors	40
Table 21 : List of Capturing Errors	41

6.2 List of Figures

Figure 1: Overview - Capturing Process	6
Figure 2: Ring Buffer Mode State Diagram	7
Figure 3: Capture File Overview - V1.3 and later	8

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 Ges.f.Systemaut. mbH
Shanghai Representative Office
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 - 110 025
Phone: +91 9810269248
E-Mail: info@hilscher.in

Italy

Hilscher Italia srl
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.
Suwon-Si, 443-810
Phone: +82-31-204-6190
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