Programming Reference Guide

netX Diagnostic and Remote Access

Fundamentals

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC090703PR03EN | Revision 3 | English | 2019-08 | Released | Public

Table of contents

1 Introduction

1.1 About this document

netX Diagnostic and Remote Access describes the possibilities to connect host systems to a netX-based target system or a remote workstation running a netX-based hardware.

Goal of the '*netX Diagnostic and Remote Access*' services is to provide a standard diagnostic interface for netX-based systems via common physical connections (serial, USB, Ethernet) in combination with standard access functions (cifX API / rcX data packages) and the possibility to use the target connection for runtime data access and data exchange between the host and the target.

The complete package consists of several software modules and is based on the '*Hilscher Transport Mechanism*'.

This fundamentals document is the introduction of the diagnostics and remote access services, giving a general overview and describes the containing software modules, data structures and underlying communication layers.

1.2 List of revisions

Rev	Date	Name	Chapter	Revision
3	2019-08-26	MS, MT, RM, ALM	all	usTransaction/changed ulReserved to usReserved in all header definitions added.
			2.1s	Overview figure updated.
			2.4, 2.4.2, 3.2.1	Transfer types 0x300 (INX), 0x400 (netANALYZER), and 0x500 (netPLC) added. Section *Device-specific data transfer types* added.
			6	Marked Marshaller frame sequence as unused Table 27: netXMarshaller Method ID definitions updated.
			6.3.11	Section MARSHALLER_SYSDEV_METHODID_RESETEX added.

Table 1: List of revisions

1.3 Terms, abbreviations and definitions

Term	Description
API	**A**pplication **P**rogramming **I**nterface
CDC	**C**ommunication **D**evice **C**lass (USB specification)
cifX	**C**ommunication **I**nter**f**ace based on net**X**
CMD	Command
comX	**Co**mmunication **M**odule based on net**X**
DPM	**D**ual **P**orted **M**emory
netX	Next Generation of Communication Controllers
ODM	Online Data Manager
OS	Operating System
rcX	**r**eal-time **c**ommunication system net**X**
SDK	**S**oftware **D**evelopment **K**it
xC	Communications channel on the netX chip (= xPEC + xMAC)
xMAC	Medium Access Control component on the netX chip
xPEC	Protocol Execution Controller component on the netX chip

Table 2: Terms, abbreviations and definitions

1.4 References to documents

This document based on the following specifications and manuals:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX Dual-Port Memory Interface, Revision 16, DOC060302DPM16EN, English, 2019.

[2] Hilscher Gesellschaft für Systemautomation mbH: Driver Manual, cifX Device Driver, Windows 2000/XP/Vista/7/8/10, Revision 27, DOC060701DRV27EN, English, 2019.

[3] Hilscher Gesellschaft für Systemautomation mbH: Programming reference guide, CIFX API, Revision 7, DOC121201PR07EN, Englisch, 2019.

[4] ODM V3 Specification, Rev. 3, 2006-10-18.

Table 3: References to documents

2 Architectural basis

The '*netX Diagnostic and Remote Access*' covers various functionalities concerning a host and netX communication and the transfer of diagnostic and run-time data between the systems.

Functionalities

- Data Transport and Representation
  The physical data transport is based on the '*Hilscher Transport Mechanism*', describing a physical connection independent handling of a user data transport between a host and a target (connection independent)
- Connectivity to different transport medias (USB / serial / Ethernet)
- Source code of the target implementation portable to so called '*Remote Systems*', running a netX target with different operating system (OS independent C-Sources)
- Hardware connection independent access to the target system via rcX data packages or remote access via cifX API functions

2.1 Overview

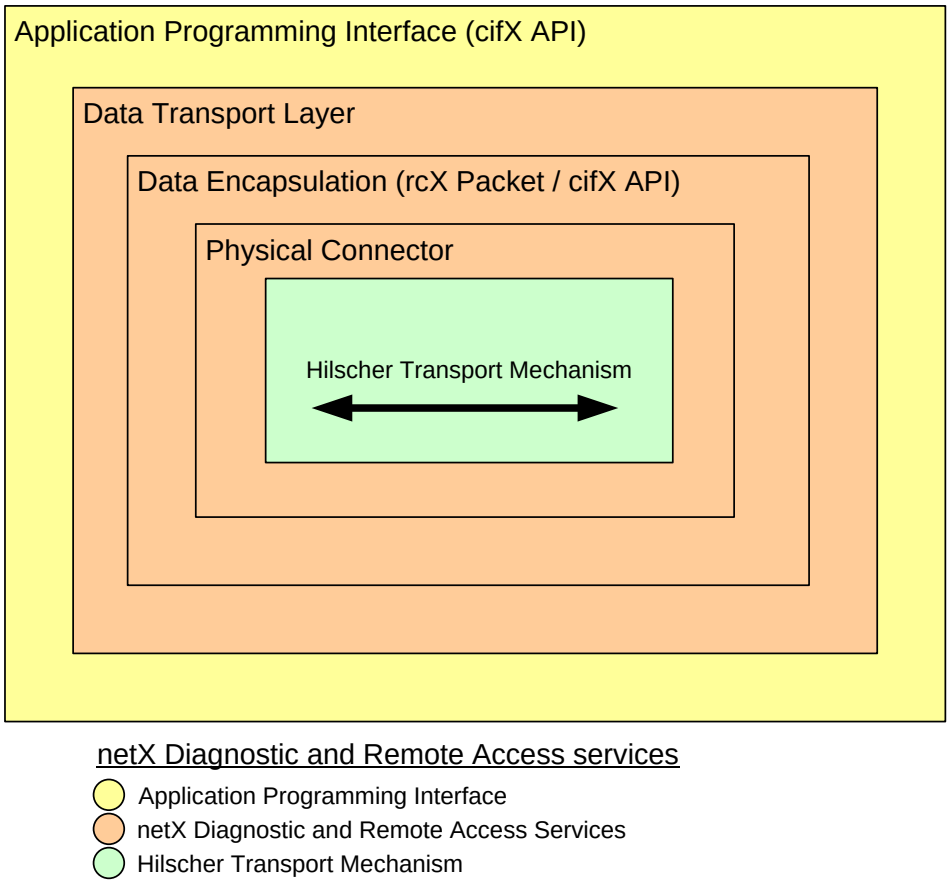

Figure 1: Overview

2.2 Software structure

The software consists of two parts, the so called 'Host-Side' and 'Target-Side' part.

netXMarshaller describes the administration and controlling function inside the remote access services.

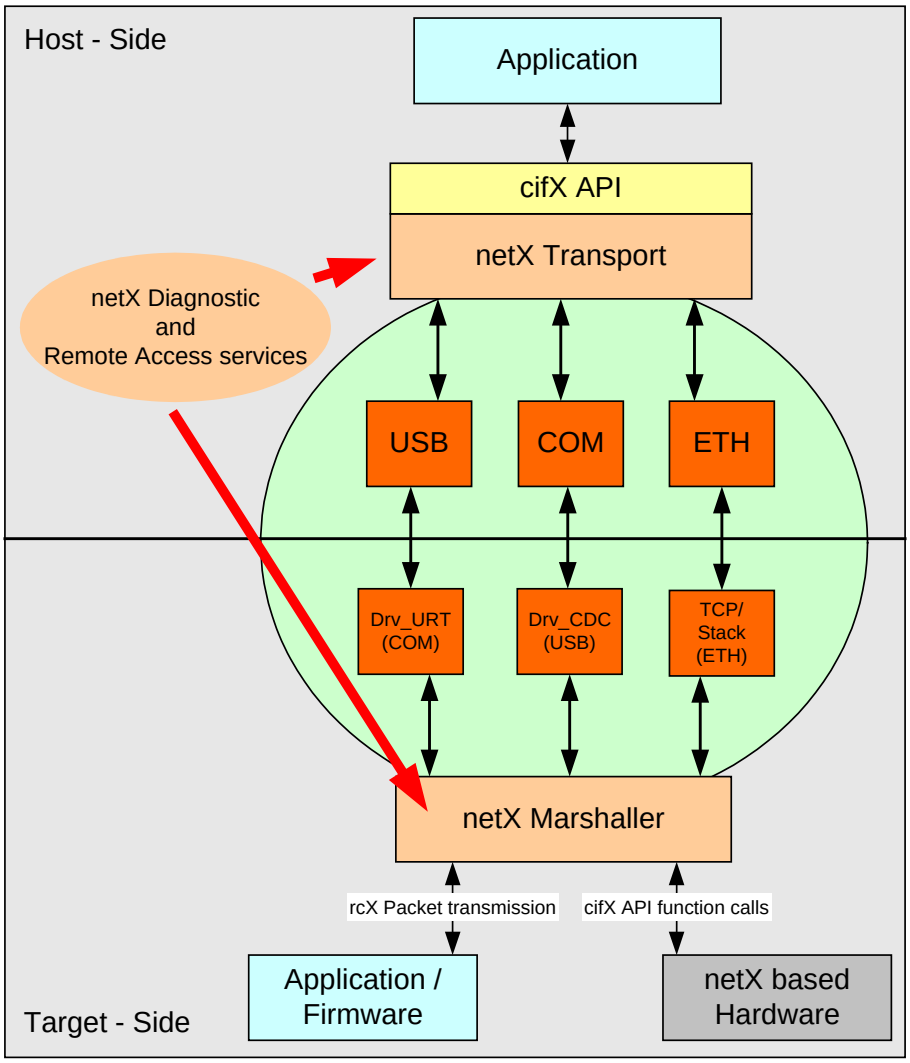

Figure 2: Software structure

The **Host-Side** is created for systems based on Microsoft Windows, while the **Target-Side** is implemented in ANSI-C format and the source code is designed to be usable on different operating systems.

Both sides are described in separate programming reference manuals.

Internal block diagram

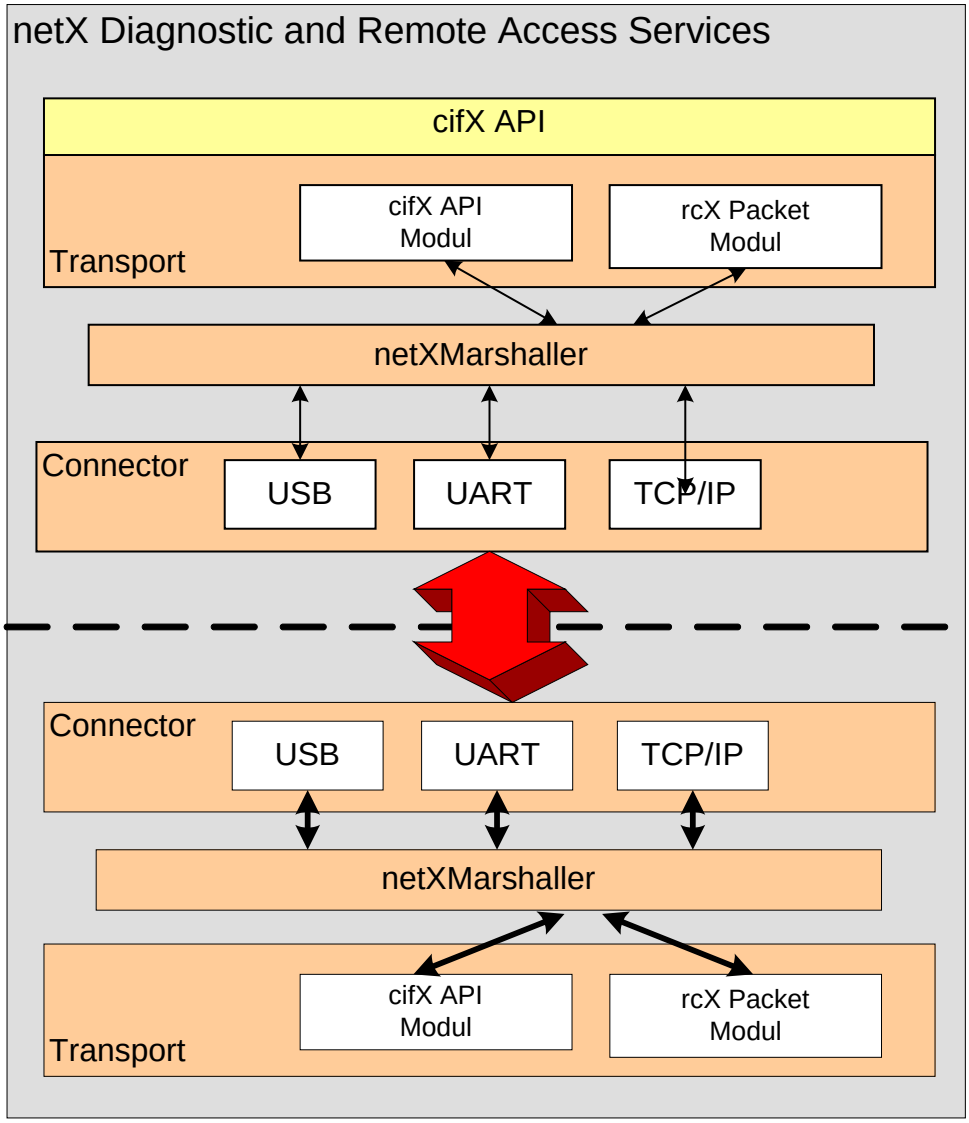

Figure 3: Internal block diagram

Internal data flow

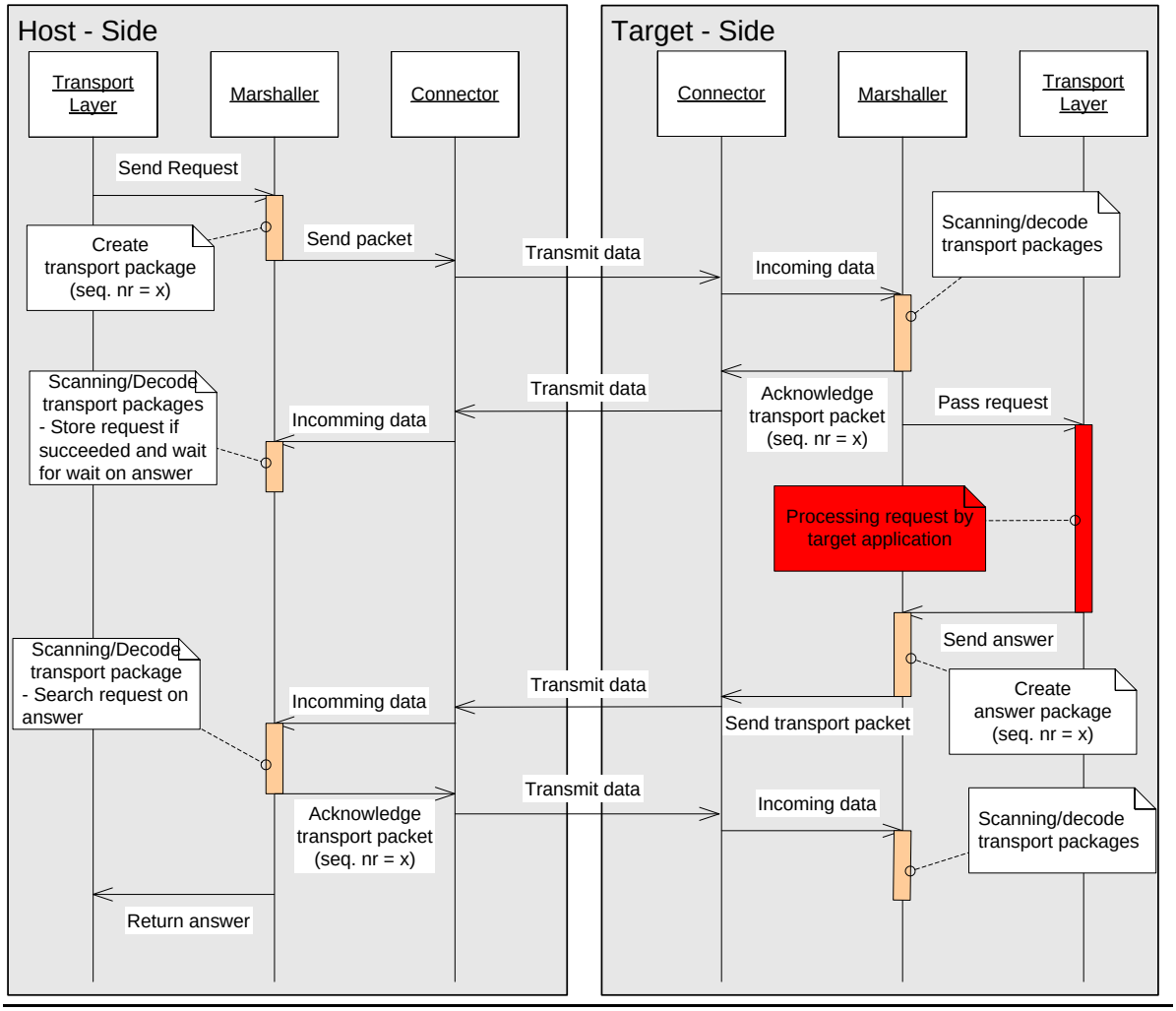

Figure 4: Internal data flow

2.3 cifX Application Interface

The **netX Diagnostic and Remote Access** services are accessibly by an application via the cifX API. For a description, see reference [3].

2.4 Data transfer types

2.4.1 Standard data transfer types

User data are transferred either via the standard rcX packet transfer mechanism or by remote cifX API function calls, processed on the target system.

The target system defines which type of data transfer is used. The netXMarshaller internally handles the transfer depending on the transfer type code given in the frame header.

This handling between the source and the target is transparent to the application which still uses the cifX API functions independent from the internal transfer.

The following transfer types are built into the netXMarshaller library:

- rcX Packet transfer
- cifX API remote procedure calls

2.4.2 Device-specific data transfer types

The following transfer types have been defined in order to support the communication requirements of specific devices that provide different or additional APIs for access from remote:

- INX API remote procedure calls: The INX communication API is provided by devices based on the industrialNETworX development platform.
- netANALYZER API remote procedure calls: This communication API provides specific functionality for the use with netANALYZER devices.
- netPLC API remote procedure calls: This communication API provides specific functionality for the use with netPLC devices.

The netXMarshaller supports these transfer types if the corresponding transfer component is built into the device firmware and is configured and registered with the netXMarshaller core part.

3 Hilscher Transport Mechanism

The **Hilscher Transport Mechanism** specifies a low level data transmission protocol of user data, independent of the physical connection, the contained data and usable for all kinds of user data streams.

The transported data are prefixed by a special transport header, called **Hilscher Transport Header** which is automatically prefixed by the transmission layer sending the data. The transport header contains the necessary information like a sequence number and a checksum to provide data consistency. The data transfer is protected by a hand-shake handling.

As the transport mechanism does not interpret the contained data, the upper layer protocols (transport layers) are responsible to insert own sequencing information, to be able to correctly assign requests and answers. And they are also responsible to provide a predefined transport data type ('*Data Type*') describing the contained user data.

The transport ID is used by the receiving transport header interpreter to determine which transport layer, on the target, shall receive the data. The interpretation and processing of the user data is the task of the transport layers itself.

Note: Each transport packet will get a unique sequence number during transmission. Receiver and transmitter using independent sequence numbers for their transmission, so sequence number can only be used to detect the loss of data packets but cannot be used to assign answers to request.

3.1 Data encapsulation

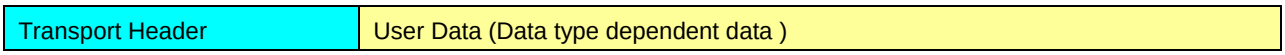

Figure 5: Data encapsulation

3.2 Transport Header definition

The following table shows the definition of the *Hilscher Transport Header*:

Element	Data type	Description
ulCookie	DWORD	Telegram Cookie determining the start of a data packet Value = **0xA55A5AA5**
ulLength	DWORD	Length of packet data following the header
usChecksum	WORD	CRC-16 Checksum of the packet data, following the header **0** = no checksum available
usDataType	WORD	Type of packet data **HIL_TRANSPORT_TYPE_RCX_PACKET** **HIL_TRANSPORT_TYPE_MARSHALLER** **HIL_TRANSPORT_TYPE_ACKNOWLEDGE** **HIL_TRANSPORT_TYPE_KEEP_ALIVE**
bDevice	BYTE	Device number of the receiver **0** = local device
bChannel	BYTE	Communication channel number of addressed device (*bDevice*) **0xFF** = System channel **0..n** = Communication channel number
bSequenceNr	BYTE	Header sequence number Used to assign requests to acknowledge and answer packages. Incremented by each transmission
bState	BYTE	State of the packet
usTransaction	WORD	Transaction ID **0** not supported **0x0001.-.0x7FFF** Host request package **0x8000 - 0xFFFF** Device request package **Note:** The highest bit is reserved for device request
usReserved	WORD	Reserved for future use
Package Data		
-/-	-/-	Packet data - Content defined by *usDataType* - Length in bytes defined by *ulLength*

Table 4: Transport Header definition

3.2.1 Transport data types

The transport mechanism has predefined transport data types to describe the user data contained in a transport package. '*usDataType*' is used to distinguish the data.

Standard data types

Data type	Value	Description
HIL_TRANSPORT_TYPE_RCX_PACKET	0x100	Data contains rcX packet
HIL_TRANSPORT_TYPE_MARSHALLER	0x200	Data contains a cifX API function call
HIL_TRANSPORT_TYPE_ACKNOWLEDGE	0x8000	Acknowledge packet

Table 5: Standard data types

Feature supported data types

Data type	Value	Description
(administration command area)	0x00 - 0xFF	Reserved for administration commands
HIL_TRANSPORT_TYPE_KEEP_ALIVE	0xFFFF	Keep Alive packet

Table 6: Feature data types

Device-specific data types

Data type	Value	Description
HIL_TRANSPORT_TYPE_INX	0x300	Data contains an INX API function call
HIL_TRANSPORT_TYPE_NETANALYZER	0x400	Data contains a netANALYZER API function call
HIL_TRANSPORT_TYPE_NETPLC	0x500	Data contains a netPLC API function call

Table 7: Device-specific data types

3.2.2 State definitions

The state field is used in an acknowledge packet, to signal the state of the actual transmission. The following states are defined:

State definition	Value	Description
HIL_TRANSPORT_STATE_OK	0x00	No error
HIL_TSTATE_CHECKSUM_ERROR	0x10	Checksum did not match
HIL_TSTATE_LENGTH_INCOMPLETE	0x11	Length incomplete - Error waiting for packet to be received completely
HIL_TSTATE_DATA_TYPE_UNKNOWN	0x12	Unknown data type - Device is not able to handle this data type.
HIL_TSTATE_DEVICE_UNKNOWN	0x13	Unknown device number given
HIL_TSTATE_CHANNEL_UNKNOWN	0x14	Unknown channel number given
HIL_TSTATE_SEQUENCE_ERROR	0x15	Sequence number out of sync
HIL_TSTATE_BUFFEROVERFLOW_ERROR	0x16	Buffer overflow
HIL_TSTATE_RESOURCE_ERROR	0x17	Out of internal resources
HIL_TSTATE_KEEP_ALIVE_ERROR	0x20	Keep Alive communication identifier was incorrect

Table 8: State definitions

3.3 Keep Alive

The '*Keep Alive*' mechanism is an extension of the '*Hilscher Transport Mechanism*' and used to monitor active target connections. If no user data are transferred for a pre-defined time, the host starts to send supervising data packages to keep the connection active. If the target does not answer anymore to the '*Keep Alive*' packages, a target connection closed is signaled.

The '*Keep Alive'* mechanism is separated in two parts. During connection establishment the target signals if '*Keep Alive*' is supported. In this case the '*Keep Alive*' mechanism starts to send one '*Keep Alive*' request using a communication identifier of 0. This instructs the target to answer with a new communication identifier, valid for the actual connection. If the target is requested with a communication identifier not equal to 0 or equal to the actual active identifier, the request will be rejected with an error state (HIL_TSTATE_KEEP_ALIVE_ERROR).

3.3.1 First request

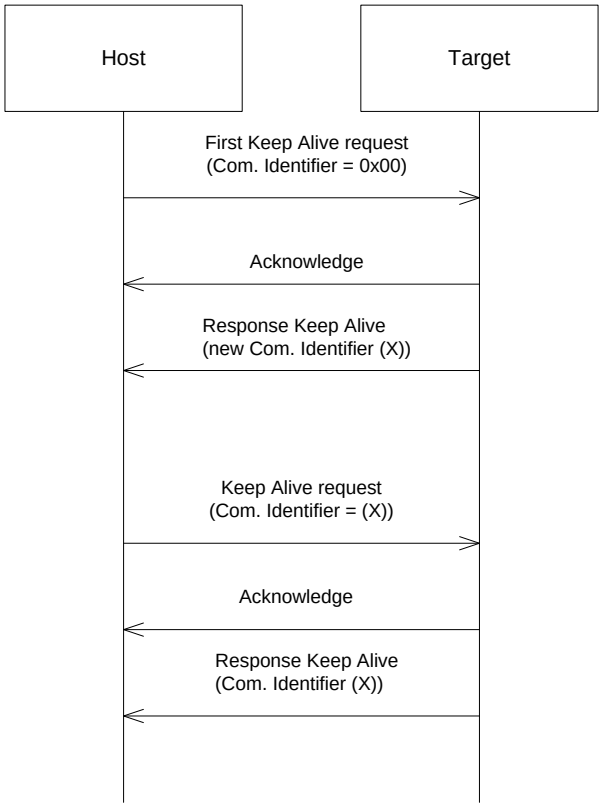

Figure 6: Keep Alive - First request

3.3.2 Continuing Keep Alive

After the initialization of the '*Communication Identifier*', this identifier needs to be used in all further requests. The communication is solely orientated by this identifier.

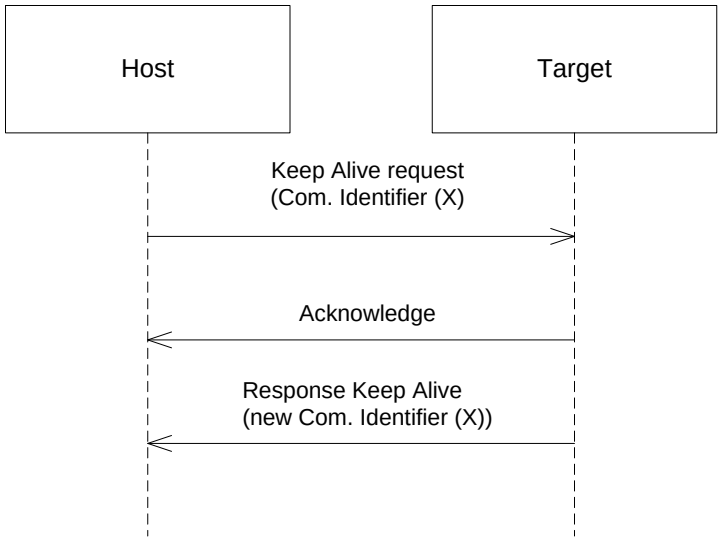

Figure 7: Keep Alive - Continuing

3.3.3 Faulty request

If the communication partner sends a request with a communication identifier unequal to the active identifier (not 0), the request will be rejected with an error state. This could happen if the host is disconnected from one target and connected to another target while holding an active connection.

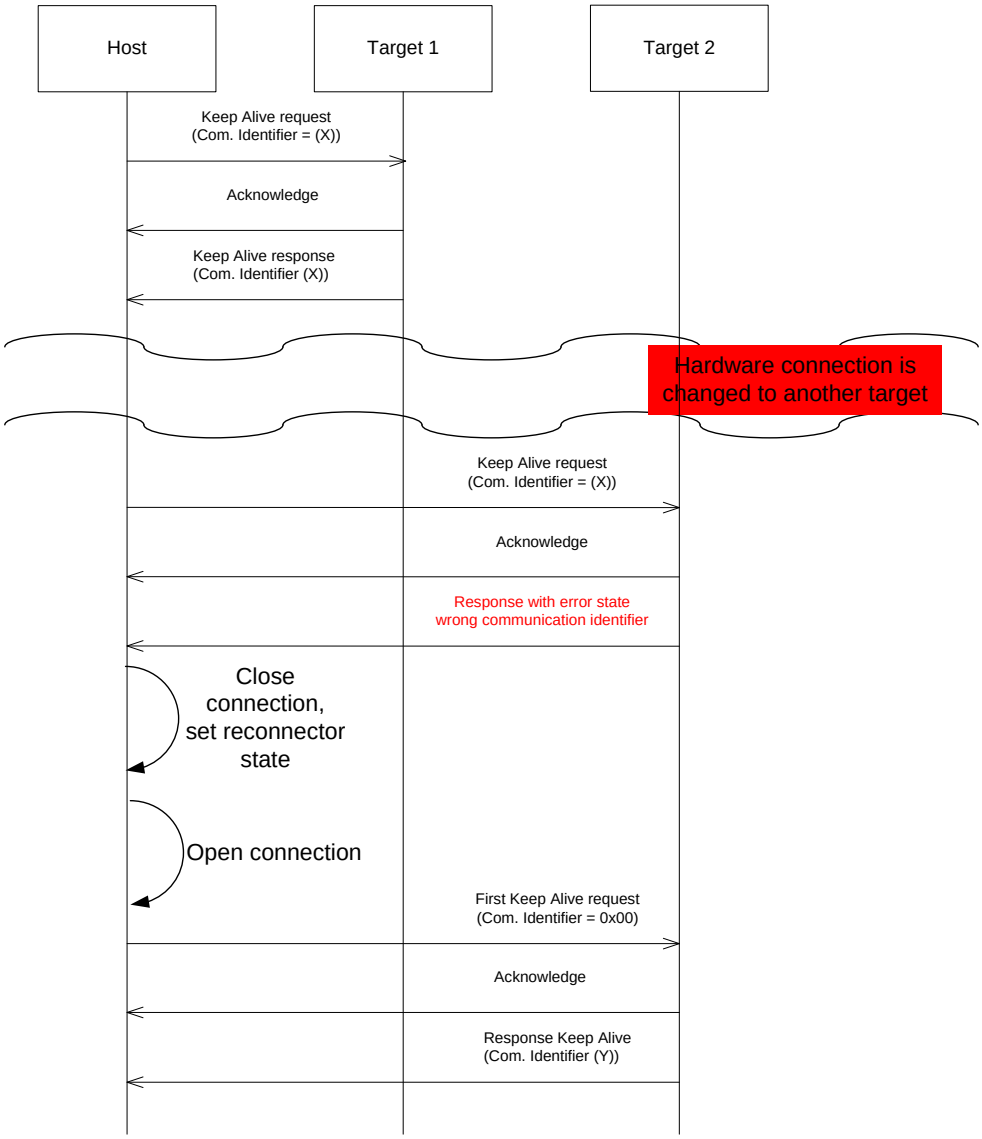

Table 9: Keep Alive - Faulty request

3.3.4 Creating an Keep Alive request

The Keep Alive request is send every time, no other packet is active (communication idle). The keep alive request will be acknowledged with a HIL_TRANSPORT_TYPE_ACKNOWLEDGE packet, like each other packet.

Hilscher Transport Header (HIL_TRANSPORT_HEADER)		
Variable	**Type**	**Description**
ulCookie	UINT32	Unique identification for data packet synchronization 0xA55A5AA5
ulLength	UINT32	Length of user data
usCheckSum	UINT16	Checksum of the packet data
usDataType	UINT32	**HIL_TRANSPORT_TYPE_KEEP_ALIVE**
bDevice	UINT8	Device number
bChannel	UINT8	Channel number
bSequenceNr	UINT8	Sequence number
bState	UINT8	Transmission state **0** = default
usTransactionID	UINT16	Transaction ID
usReserved	UINT16	Unused/Reserved for later use
Package data		
ulComID	UINT32	Communication identifier, unique identification of the communication partner. **0x00000000** = Request for a new identifier

Table 10: Keep Alive request

3.3.5 Creating an acknowledgement

Each command must be answered by the target system. An answer never includes data and is only used to signal the transmitter a correct transmission.

Hilscher Transport Header (HIL_TRANSPORT_HEADER)		
Variable	**Type**	**Description**
ulCookie	UINT32	Unique identification for data packet synchronization 0xA55A5AA5
ulLength	UINT32	Length of user data 0 for an acknowledge packet
usCheckSum	UINT16	Checksum of the user data 0 = no user data
usDataType	UINT32	**HIL_TRANSPORT_TYPE_ACKNOWLEDEG =** 0x8000
bDevice	UINT8	Original device number unchanged
bChannel	UINT8	Original channel number unchanged
bSequenceNr	UINT8	Original sequence number unchanged
bState	UINT8	Transmission/Target state != 0 defines an error
usTransaction	UINT16	Transaction ID
usReserved	UINT16	Unused/Reserved for later use

Table 11: Keep Alive acknowledge

3.3.6 Keep Alive response

This is a response to a keep alive request.

Hilscher Transport Header (HIL_TRANSPORT_HEADER)		
Variable	**Type**	**Description**
ulCookie	UINT32	Unique identification for data packet synchronization 0xA55A5AA5
ulLength	UINT32	Length of the packet data in bytes
usCheckSum	UINT16	Checksum of the packet data
usDataType	UINT32	HIL_TRANSPORT_TYPE_KEEP_ALIVE
bDevice	UINT8	Original device number - unchanged
bChannel	UINT8	Original channel number - unchanged
bSequenceNr	UINT8	Original sequence number - unchanged
bState	UINT8	Transmission/Target state != 0 defines an error
usTransaction	UINT16	Transaction ID
usReserved	UINT16	Unused/Reserved for later use
Package data		
ulComID	UINT32	Communication identifier, unique identification of the communication partner.

Table 12: Keep Alive response

3.3.7 Host functionality

The '*Keep Alive*' mechanism is implemented in the '*Transport Layer*'. The netXMarshaller is responsible to start the mechanism if a connection to a target is opened. Closing a connection will also stop the mechanism. During the start, a first request is sent to the target to retrieve a valid connection ID, needed for further connection monitoring. The keep alive management, than begins to check the transmission. If no transmission occurs certain period of time, it begins to send '*Keep Alive*' packages. A lost connection (either no or a wrong response) will be signaled to the netXMarshaller which handles the further close procedure processing.

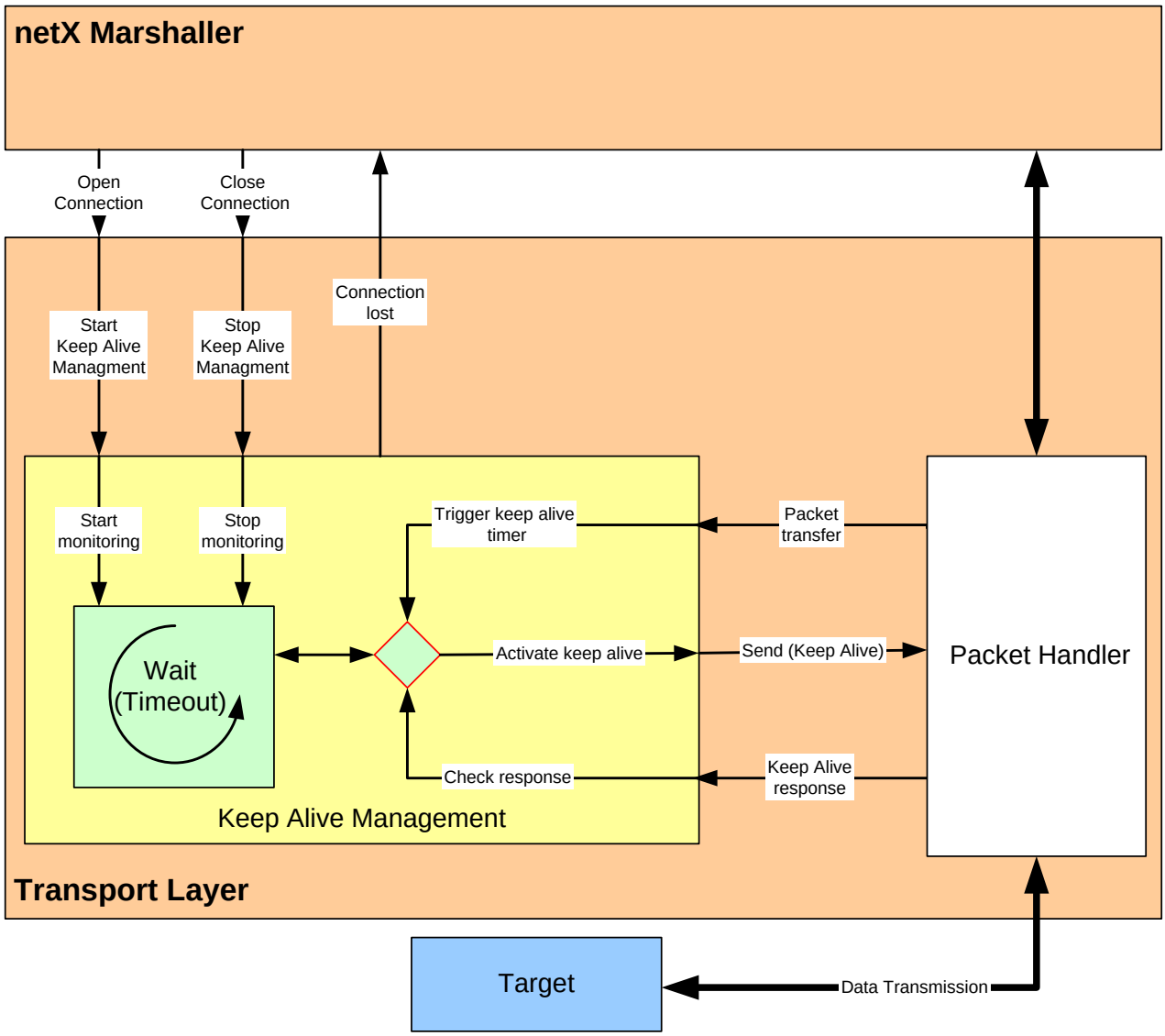

Figure 8: Keep Alive host functionality

3.3.8 Target functionality

Every time a request with a zero identifier is received, the target has to create a new communication identifier. This identifier is stored for subsequent keep alive requests. If a wrong identifier is received, the target rejects the request using the error state HIL_TSTATE_KEEP_ALIVE_ERROR.

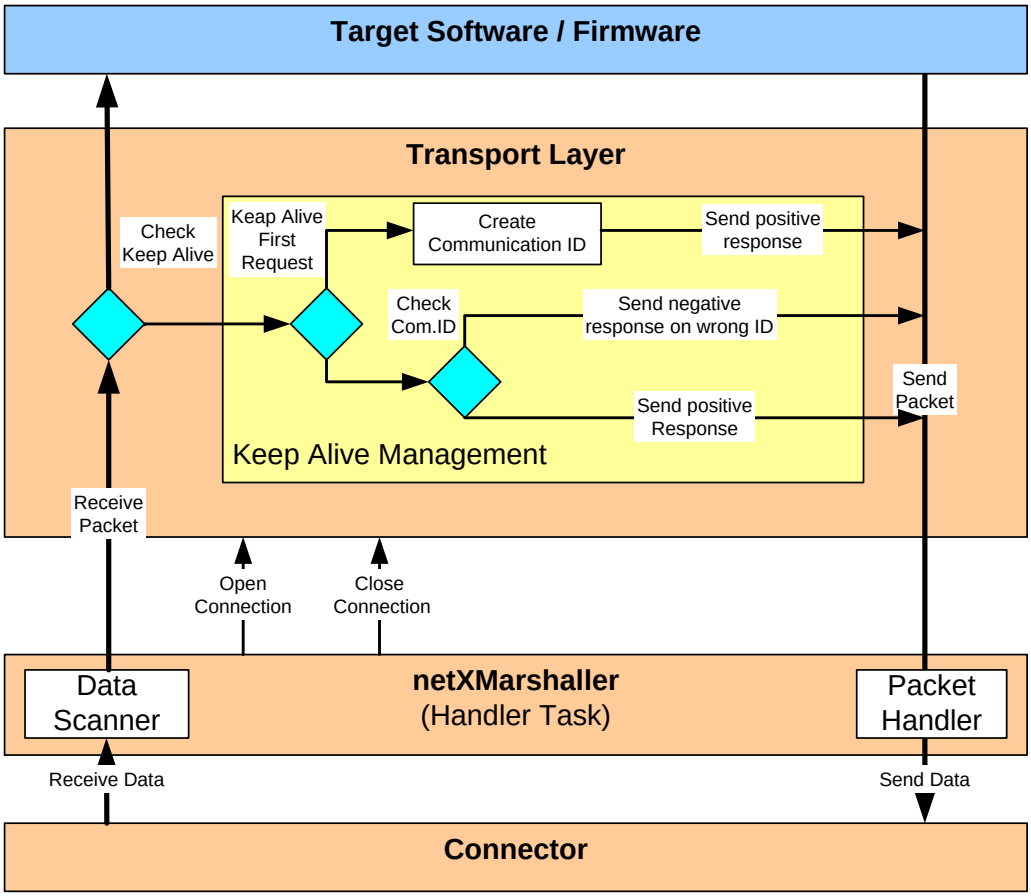

Figure 9: Keep Alive target functionality

4 netXMarshaller

netXMarshaller is the controlling functions inside the *'netX Diagnostic and Remote Access*' services. The task of the marshaller is the administration and controlling of the target connections and user data decoding and encoding.

Administration of the target hardware is done via defined commands and a setup handling during the creation of a target link.

Function Overview

- Administration and controlling of the target connections
- User data encoding and decoding

4.1 Target Administration Commands

It is necessary for the host to be able to detect the number of supported devices/channels, the supported data types and possible features. This interface must be usable on all servers to be able to keep the protocol device independent.

The following chapter defines the available administration commands and the recommended fallback behavior of the host if a device does not support these functions.

The following table shows all currently defined administration data types:

Data type	Value	Description
HIL_TRANSPORT_TYPE_QUERYSERVER	0x0000	Query basic server information
HIL_TRANSPORT_TYPE_QUERYDEVICE	0x0001	Query device specific information Depends on usDataType and is currently only defined for HIL_TRANSPORT_TYPE_RCX_PACKET

Table 13: Administration data types

Note: The first command the host is expected to send is *HIL_TRANSPORT_TYPE_QUERYSERVER*.

4.2 Querying basic server information

This command is used to query available services and supported features from the target (remote server). This function should be called by any host to determine which interfaces are available. If the command is not supported by the target, the host should use the defined fallback mechanism described in section Fallback behaviour on page 32.

The request consists of an empty transport packet with the *usDatatype* field set to '*HIL_TRANSPORT_TYPE_QUERYSERVER*'.

Response data

Element	Data type	Description
ulStructVersion	DWORD	Version of the following structure (this document describes version 1)
szServerName	char[32]	Server name, zero terminated string
ulVersionMajor	DWORD	Major version of server component
ulVersionMinor	DWORD	Minor version of server component
ulVersionBuild	DWORD	Builder number of server component
ulVersionRevision	DWORD	Revision of the server component
ulFeatures	DWORD	Bit field of supported features
ulParallelServices	DWORD	Number of parallel services the device can handle
ulBufferSize	DWORD	Maximum number of bytes, including the *Transport Header,* the server can handle (valid for results and requests)
usDatatypesCnt	WORD	Number of following supported data types
ausDatatypes	WORD[ulDatatypesCnt]	Supported data type code. **Note:** Mandatory data types like HIL_TRANSPORT_TYPE_ACKNOWLEDGE and QUERY_SERVER are not listed

Table 14: Administration - Query Server Information result

ulFeatures bit 0..15

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
															Keep Alive Supported	
Reserved, do not use (set to zero)																

ulFeatures bit 16..31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved, do not use (set to zero)																

Example

Query server request

ulCookie	0xA55A5AA5
ulLength	0
usChecksum	0
usDatatype	0
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0

Query server result

ulCookie	0xA55A5AA5					
ulLength	42					
usChecksum	0					
usDatatype	0					
bDevice	0					
bChannel	0					
bSequenceNr	n					
bState	0					
usTransactionID	m					
ulReserved	0					
ulStructVersion	1					
szServerName	n	e	t	l	C	\0
ulVersionMajor	1					
ulVersionMinor	0					
ulVersionBuild	0					
ulVersionRev	0					
ulParallelServices	1					
ulBufferSize	2400					
usDatatypesCnt	3					
aulDatatypes[0]	0x0001					
aulDatatypes[1]	0x0200					
aulDatatypes[2]	0xFFFF					

Figure 10: Query server information - Example

4.3 Query device information

The query device information depends on the data type. For cifX API commands this query is not needed, as it is available through the interface itself. This method should only be used for RCX_PACKET based transport. The basic query command includes the data type and the requested option.

Request data

Element	Data type	Description
usDatatype	WORD	Data type the request is made for
ulOption	DWORD	Requested option

Table 15: Administration - Query Device Information - Basic Request

Query options

Option	Value	Description
QUERY_DEVICE_OPT_DEVICECNT	0	Get the total number of devices (Should be send with bDevice=bChannel=0, as these devices exist on every server)
QUERY_DEVICE_OPT_CHANNELCNT	1	Get the total number of channels on a device bDevice = device to be queried
QUERY_DEVICE_OPT_DEVICEINFO	2	Get device information bDevice = device to be queried
QUERY_DEVICE_OPT_CHANNELINFO	3	Get channel information bDevice = device to be queried bChannel = channel to be queried

Table 16: Administration - Query device information - Options list

4.3.1 Query number of devices with HIL_TRANSPORT_TYPE_RCX_PACKET

The following chapter describes the commands and responses used to query all existing communication partners on the target.

Response data

Element	Data type	Description
usDatatype	WORD	Data type from request
ulOption	DWORD	Option from request
ulError	DWORD	Error code for the request (see Table 21) **Note:** If *ulError* is set, the following data will be suppressed.
ulDeviceCnt	DWORD	Number of supported devices

Table 17: Administration - Query Number of Devices - Result

Example

Query device count request

ulCookie	0xA55A5AA5
ulLength	6
usChecksum	0
usDatatype	1
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	0

Query device count result

ulCookie	0xA55A5AA5
ulLength	14
usChecksum	0
usDatatype	0
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	0
ulError	0
ulDeviceCnt	1

Figure 11: Administration - Query number of devices - Example

4.3.1.1 Query number of Communication Channels

Response data

Element	Data type	Description
usDatatype	WORD	Data type from request
ulOption	DWORD	Option from request
ulError	DWORD	Error code for the request (see Table 21) **Note:** If *ulError* is set, the following data will be suppressed.
ulChannelCnt	DWORD	Number of supported channels

Table 18: Administration - Query Number of Communication Channels - Result

Example

Query channel count request

ulCookie	0xA55A5AA5
ulLength	6
usChecksum	0
usDatatype	1
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	1

Query channel count result

ulCookie	0xA55A5AA5
ulLength	14
usChecksum	0
usDatatype	0
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	1
ulError	0

Figure 12: Administration - Query number of Communication Channel - Example

4.3.1.2 Query device information

Response data

Element	Data type	Description
usDatatype	WORD	Data type from request
ulOption	DWORD	Option from request
ulError	DWORD	Error code for the request (see Table 21) **Note:** If *ulError* is set, the following data will be suppressed.
ulDeviceNr	DWORD	Device number
ulSerialNr	DWORD	Serial number
usMfg	WORD	Manufacturer code
usProdDate	WORD	Production date
ulLicenseFlags1	DWORD	License flags 1
ulLicenseFlags2	DWORD	License flags 2
usLicenseID	WORD	License ID
usLicenseFlags	WORD	License flags
usDeviceClass	WORD	Device class
bHwRevision	BYTE	HW Revision
bHwCompatibility	BYTE	HW compatibility

Table 19: Administration - Query device information - Result

Example

Query device info request

ulCookie	0xA55A5AA5
ulLength	6
usChecksum	0
usDatatype	1
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	2

Query device info result

ulCookie	0xA55A5AA5
ulLength	38
usChecksum	0
usDatatype	0
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	2
ulError	0
ulDeviceNr	1250100
ulSerialNr	20029
usMfg	1
usProdDate	507
ulLicenseFlags1	0x0000007F
ulLicenseFlags2	0x00000001
usLicenseID	0
usLicenseFlags	0
usDeviceClass	0
bHwRevision	0
bHwCompatibility	0

Figure 13: Query device information - Example

4.3.1.3 Query channel information

Response data

Element	Data type	Description
usDatatype	WORD	Data type from request
ulOption	DWORD	Option from request
ulError	DWORD	Error code for the request (see Table 21) **Note:** If *ulError* is set, the following data will be suppressed.
usCommClass	WORD	Communication class
usProtocolClass	WORD	Protocol Class
usConformanceClass	WORD	Protocol Conformance class
szFWName	CHAR[]	Firmware name as zero terminated string
ulFWMajor	DWORD	Firmware version (major)
ulFWMinor	DWORD	Firmware version (minor)
ulFWBuild	DWORD	Firmware version (build)
ulFWRev	DWORD	Firmware version (revision)
usFWYear	WORD	Build data of firmware (year)
bMonth	BYTE	Build data of firmware (month)
bDay	BYTE	Build data of firmware (day)

Table 20: Administration - Query channel information - Result

Example

Query channel info request

ulCookie	0xA55A5AA5
ulLength	6
usChecksum	0
usDatatype	1
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	3

Query channel info result

ulCookie	0xA55A5AA5
ulLength	45
usChecksum	0
usDatatype	0
bDevice	0
bChannel	0
bSequenceNr	n
bState	0
usTransactionID	m
ulReserved	0
usDatatype	0x100
ulOption	3
ulError	0
usCommClass	3
usProtocolClass	0x13
usConformance	0
szFWName	P R O F I B U S \0
ulFWMajor	2
ulFWMinor	0
ulFWBuild	10
ulFWRev	0
usFWYear	2008
bFWMonth	5
bFWDay	11

Figure 14: Query channel information - Example

4.4 Fallback behaviour

The following diagram defines the host fallback behaviour, if commands are not available on the target (e.g. older firmware versions).

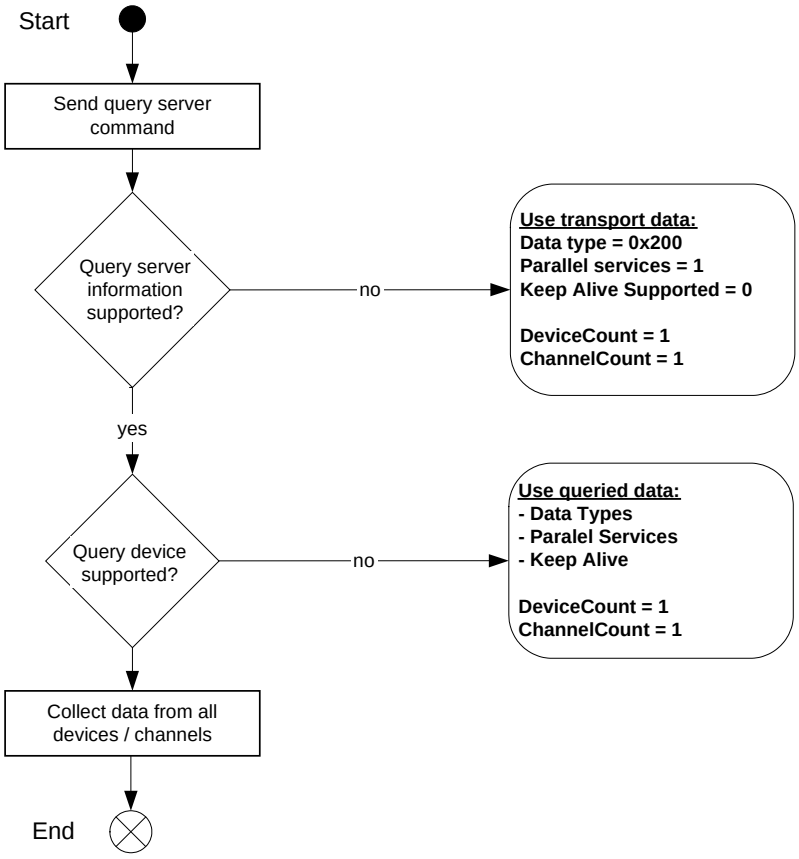

Figure 15: Fallback behaviour

4.5 Error codes

Definition	Value	Description
HIL_ADMIN_S_OK	0x00000000	Request was successfully executed
HIL_ADMIN_UNSUPPORTED	0x80000000	Request is unsupported
HIL_ADMIN_DEVICE_NOTEXISTING	0x80000001	Addresses device is not existent *bDevice* is invalid
HIL_ADMIN_CHANNEL_NOTEXISTING	0x80000002	Channel is not existing on the device *bChannel* invalid

Table 21: Error codes for administration commands

5 rcX Packet Transfer

The rcX packet transfer is used to send rcX packets to a rcX-based target system. The rcX packet (header and data) is placed directly in the user data area of the transport packet.

rcX Packet structure			
Area	**Variable**	**Type**	**Description**
Data header			
tHeader	`ulDest`	UINT32	Destination Queue Handle
	`ulSrc`	UINT32	Source Queue Handle
	`ulDestId`	UINT32	Destination Queue Reference
	`ulSrcId`	UINT32	Source Queue Reference
	`ulLen`	UINT32	Packet Data Length (in Bytes, without the header)
	`ulId`	UINT32	Packet Identification as Unique Number
	`ulState`	UINT32	Status
	`ulCmd`	UINT32	Command
	`ulExt`	UINT32	Extension
	`ulRout`	UINT32	Routing Information (internally used)
Package Data			
tData	packet depending		User data

Table 22: rcX Packet - Basic structure

Note: For a detailed description of rcX packet, see reference [1].

Incoming rcX packets on a target device shall be handled in the same way as if they have been received via a DPM mailbox. For this handling the target device needs information about the receiving device / channel. The transport header offers elements to address the device and the channel.

The following table shows the used transport header elements:

Header element	**Description / Mapping**
bDevice	Device number on the target system that should receive the packet. **Note:** A device handled a separate cifX device
bChannel	Channel number the packet is send to

Table 23: rcX Packet Transfer (Channel/Device mapping)

5.1 Addressing example

The host wants to access a target that is controlling 2 different devices. '*Device 0*' contains 2 communication channels (communication protocols) while '*Device 1*' has no active communication channel (system device only).

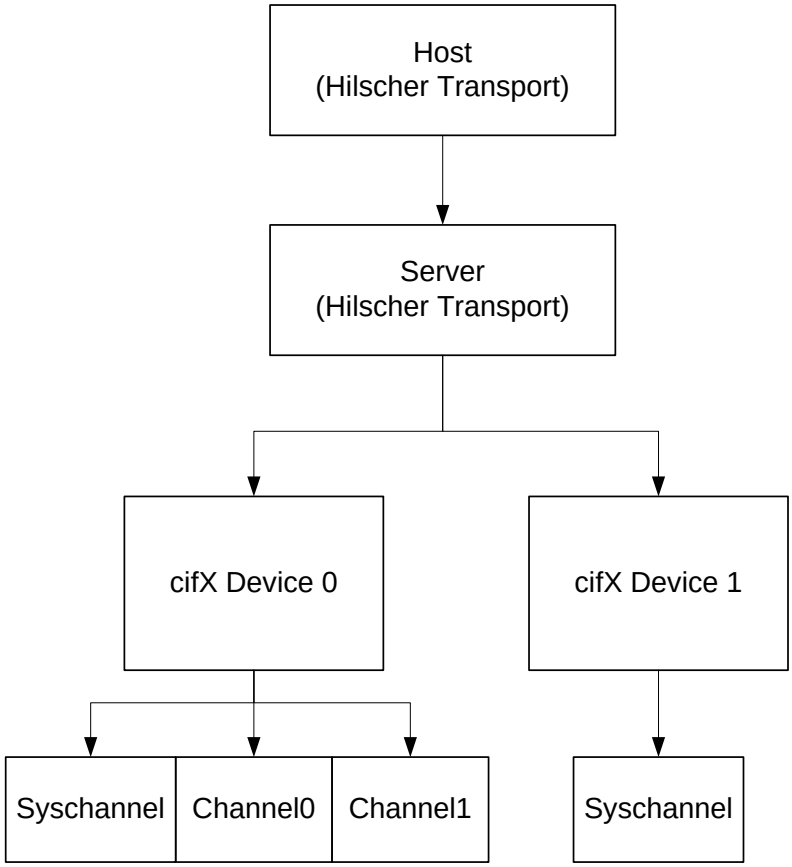

Figure 16: rcX Packet Transmission example

With this setup the host will need to set *'bDevice'* to access each of the devices and set *'bChannel'* to the proper channel value.

The following table shows all possible communications:

Header field ***bDevice***	**Header field** ***bChannel***	**Destination**
0	1	Packet will be send to cifX Device 0 / Channel 0. (AP Task of protocol stack)
0	2	Packet will be send to cifX Device 0 / Channel 1. (AP Task of protocol stack)
0	0	Packet will be send to cifX Device 0 This is the RX_SYSTEM task on this device
1	0	Packet will be send to cifX Device 1 This is the RX_SYSTEM task on this device

Table 24: rcX Packet - Addressing example

6 cifX API Data Transfer

For netX based devices, offering an rcX SHM-API or for remote system with a DPM access to netX based devices, cifX API functions calls can be '*Marshalled*' from the host to the target.

Such functions are executed on the target and the answers are returned back to the host. This allows a host application to use the cifX API functions to access netX devices which are installed in the host system (DPM access) or which are connected remotely to the host.

The cifX API function transport uses an own header definition inside the data portion of the transport header packet.

Definition of the netXMarshaller data header:

MARSHALLER_DATA_FRAME_HEADER_T		
Element	**Data type**	**Description**
ulHandle	DWORD	Handle of the *Marshaller* object
ulMethodId	DWORD	Method ID defining the function to execute (depends on the *Marshaller* object)
ulStatus	DWORD	Marshaller data frame state (see below)
ulError	DWORD	'*cifX Driver*' error code (set in a result frame) **Note:** See cifX device driver documentation for a list of possible errors.
ulSize	DWORD	Length of the following function data

Table 25: netXMarshaller Data Frame Header

ulHandle

During the creation of a target connection, the netXMarshaller, on the target, initializes the internal connection handler and creates handler objects (Marshaller Objects) for the different functionalities. A returned *ulHandle* identifies the function block and their object.

ulHandle bit 0 ... 31

Bit	**Description**
0 … 7	Object ID
8 … 15	Board Number
16 … 23	Channel Number
24 … 30	Reserved
31	Handle Valid

The handle represents one of the following objects:

ObjectID	Value	Description
MARSHALLER_OBJECT_TYPE_CLASSFACTORY	0	Used during connection establishment to create the necessary basic objects
MARSHALLER_OBJECT_TYPE_DRIVER	1	Used for driver functions: cifX API Functions: **xDriver....**
MARSHALLER_OBJECT_TYPE_SYSDEVICE	2	Used for system device functions cifX API Functions: **xSysdevice....**
MARSHALLER_OBJECT_TYPE_CHANNEL	3	Used for communication channel functions: cifX API Functions: **xChannel....**

Table 26: netXMarshaller Handle definitions

ulMethodId

The *ulMethodId* describes the function which is transferred or requested. The method ID is only unique in conjunction with a given *ulHandle*.

List of method IDs (*ulMethodId*) according to the object handle and cifX API function:

Method ID	Value	Description / cifX API Function Equivalent
Class Factory Object ID		
MARSHALLER_CF_METHODID_SERVERVERSION	0x00	Query the version of the netXMarshaller
MARSHALLER_CF_METHODID_CREATEINSTANCE	0x01	Create a new object
Driver Object ID		
MARSHALLER_DRV_METHODID_OPEN	0x01	xDriverOpen
MARSHALLER_DRV_METHODID_CLOSE	0x02	xDriverClose
MARSHALLER_DRV_METHODID_GETINFO	0x03	xDriverGetInformation
MARSHALLER_DRV_METHODID_ERRORDESCR	0x04	xDriverGetErrorDescription
MARSHALLER_DRV_METHODID_ENUMBOARDS	0x05	xDriverEnumBoards
MARSHALLER_DRV_METHODID_ENUMCHANNELS	0x06	xDriverEnumChannels
MARSHALLER_DRV_METHODID_OPENCHANNEL	0x08	xChannelOpen
MARSHALLER_DRV_METHODID_OPENSYSDEV	0x09	xSysdeviceOpen
System Device Object ID		
MARSHALLER_SYSDEV_METHODID_CLOSE	0x01	xSysdeviceClose
MARSHALLER_SYSDEV_METHODID_INFO	0x02	xSysdeviceInfo
MARSHALLER_SYSDEV_METHODID_RESET	0x03	xSysdeviceReset
MARSHALLER_SYSDEV_METHODID_GETMBXSTATE	0x04	xSysdeviceGetMBXState
MARSHALLER_SYSDEV_METHODID_PUTPACKET	0x05	xSysdevicePutPacket
MARSHALLER_SYSDEV_METHODID_GETPACKET	0x06	xSysdeviceGetPacket
MARSHALLER_SYSDEV_METHODID_DOWNLOAD	0x07	xSysdeviceDownload
MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE	0x08	xSysdeviceFindFirstFile
MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE	0x09	xSysdeviceFindNextFile
MARSHALLER_SYSDEV_METHODID_UPLOAD	0x10	xSysdeviceUpload
MARSHALLER_SYSDEV_METHODID_RESETEX	0x11	xSysdeviceResetEx
Communication Channel Object ID		
MARSHALLER_CHANNEL_METHODID_CLOSE	0x01	xChannelClose
MARSHALLER_CHANNEL_METHODID_DOWNLOAD	0x02	xChannelDownload
MARSHALLER_CHANNEL_METHODID_GETMBXSTATE	0x03	xChannelGetMBXState
MARSHALLER_CHANNEL_METHODID_PUTPACKET	0x04	xChannelPutPacket
MARSHALLER_CHANNEL_METHODID_GETPACKET	0x05	xChannelGetPacket
MARSHALLER_CHANNEL_METHODID_GETSENDPACKET	0x06	xChannelGetSendPacket
MARSHALLER_CHANNEL_METHODID_CONFIGLOCK	0x07	xChannelConfigLock
MARSHALLER_CHANNEL_METHODID_RESET	0x08	xChannelReset
MARSHALLER_CHANNEL_METHODID_INFO	0x09	xChannelInfo
MARSHALLER_CHANNEL_METHODID_WATCHDOG	0x10	xChannelWatchdog
MARSHALLER_CHANNEL_METHODID_HOSTSTATE	0x11	xChannelHostState

Method ID	Value	Description / cifX API Function Equivalent
MARSHALLER_CHANNEL_METHODID_IOREAD	0x12	xChannelIORead
MARSHALLER_CHANNEL_METHODID_IOWRITE	0x13	xChannelIOWrite
MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA	0x14	xChannelIOReadSendData
MARSHALLER_CHANNEL_METHODID_BUSSTATE	0x15	xChannelBusState
MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK	0x16	xChannelControlBlock
MARSHALLER_CHANNEL_METHODID_STATUSBLOCK	0x17	xChannelCommonStatusBlock
MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK	0x18	xChannelExtendedStatusBlock
MARSHALLER_CHANNEL_METHODID_USERBLOCK	0x19	xChannelUserBlock (not implemented)
MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE	0x20	xChannelFindFirstFile
MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE	0x21	xChannelFindNextFile
MARSHALLER_CHANNEL_METHODID_UPLOAD	0x22	xChannelUpload
MARSHALLER_CHANNEL_METHODID_IOINFO	0x23	xChannelInfo

Table 27: netXMarshaller Method ID definitions

ulStatus

This is the Marshaller data frame state. It contains information if the frame is a result or request frame, if a Marshaller frame sequence number is supported and a frame sequence number.

The Marshaller frame sequence number may be used to distinguish multiple identical marshaller frames (multiple frames with the same method ID) and to correctly assign multiple responses to their according requests. It is defined for future use as using bSequenceNr from the transport header is currently sufficient to assign responses to their originating requests.

ulStatus bit 0 .. 15

Bit	Description
0	0 = Result 1= Request
1	1 = Support of Marshaller frame sequence numbers **Note:** Currently unsupported/unused and defined for future use
2 … 15	Reserved

ulStatus bit 16 … 31

Bit	Description
16 … 31	Marshaller frame sequence number (supported if bit 1 is set) **Note:** Currently unsupported/unused and defined for future use

6.1 Classfactory Object - MethodeID

The class factory object is used to initiate a new connection and acquire the necessary driver object to connect to the cifX device driver API. If a NULL handle is supplied, all request will be redirected to a default class factory object.

6.1.1 MARSHALLER_CF_METHODID_SERVERVERSION

Query the version of the target netXMarshaller.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	0 or a previously created class factory object handle
ulMethodId	DWORD	MARSHALLER_CF_METHODID_SERVERVERSION
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 28: MARSHALLER_CF_METHODID_SERVERVERSION - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulVersion	DWORD	Contains the target netXMarshaller version

Table 29: MARSHALLER_CF_METHODID_SERVERVERSION - Result

6.1.2 MARSHALLER_CF_METHODID_CREATEINSTANCE

Create a new base object. Only a class factory object or a driver object can be created.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	0 or a previously created class factory object handle
ulMethodId	DWORD	MARSHALLER_CF_METHODID_CREATEINSTANCE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulObjectID	DWORD	Contains the object ID of the object to create: 0 = MARSHALLER_OBJECT_TYPE_CLASSFACTORY 1 = MARSHALLER_OBJECT_TYPE_DRIVER

Table 30: MARSHALLER_CF_METHODID_CREATEINSTANCE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulHandle	DWORD	Handle of the newly created object.

Table 31: MARSHALLER_CF_METHODID_CREATEINSTANCE - Result

6.2 Driver Object - MethodID

The driver object offers all basic cifX API driver related functions. Some functions deliver a new handle (e.g. *xChannelOpen()*).

6.2.1 MARSHALLER_DRV_METHODID_OPEN

Corresponding cifX API function: *xDriverOpen()*

The handle returned by *xDriverOpen()* is not passed to the caller. It is stored inside the server and referenced the by driver object on all further driver calls to this object.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_OPEN
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 32: MARSHALLER_DRV_METHODID_OPEN - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 33: MARSHALLER_DRV_METHODID_OPEN - Result

6.2.2 MARSHALLER_DRV_METHODID_CLOSE

Corresponding cifX API function: *xDriverClose()*

This will invalidate the stored handle acquired by *MARSHALLER_DRV_METHODID_OPEN*.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_CLOSE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 34: MARSHALLER_DRV_METHODID_CLOSE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 35: MARSHALLER_DRV_METHODID_CLOSE - Result

6.2.3 MARSHALLER_DRV_METHODID_GETINFO

Corresponding cifX API function: *xDriverGetInformation()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_GETINFO
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulSize	DWORD	*ulSize* parameter passed to *xDriverGetInformation()*

Table 36: MARSHALLER_DRV_METHODID_GETINFO - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	sizeof(DRIVER_INFORMATION)
Data		
tDriverInfo	DRIVER_INFORMATION	Structure containing the driver information

Table 37: MARSHALLER_DRV_METHODID_GETINFO - Result

6.2.4 MARSHALLER_DRV_METHODID_ERRORDESCR

Corresponding cifX API function: *xDriverGetErrorDescription()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_ERRORDESCR
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulSize	DWORD	*ulSize* parameter passed to *xDriverGetErrorDescription()*

Table 38: MARSHALLER_DRV_METHODID_ERRORDESCR - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	Size of following data in bytes
Data		
szErrorDescription	CHAR[*]	Error description string

Table 39: MARSHALLER_DRV_METHODID_ERRORDESCR - Result

6.2.5 MARSHALLER_DRV_METHODID_ENUMBOARDS

Corresponding cifX API function: *xDriverEnumBoards()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_ENUMBOARDS
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8
Data		
ulBoard	DWORD	Board number (0..n)
ulSize	DWORD	Size of the returned BOARD_INFORMATION structure

Table 40: MARSHALLER_DRV_METHODID_ENUMBOARDS - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	sizeof(BOARD_INFORMATION)
Data		
tBoardInfo	BOARD_INFORMATION	Structure containing the board information

Table 41: MARSHALLER_DRV_METHODID_ENUMBOARDS - Result

6.2.6 MARSHALLER_DRV_METHODID_ENUMCHANNELS

Corresponding cifX API function: *xDriverEnumChannels()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_ENUMCHANNELS
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulBoard	DWORD	Board number (0..n)
ulChannel	DWORD	Channel number (0..m)
ulSize	DWORD	Size of the returned BOARD_INFORMATION structure

Table 42: MARSHALLER_DRV_METHODID_ENUMCHANNELS - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	sizeof(CHANNEL_INFORMATION)
Data		
tBoardInfo	CHANNEL_INFORMATION	Structure containing the channel information

Table 43: MARSHALLER_DRV_METHODID_ENUMCHANNELS - Result

6.2.7 MARSHALLER_DRV_METHODID_OPENCHANNEL

Corresponding cifX API function: *xChannelOpen()*

The returned handle needs to be used inside the netMarshaller header to access the created channel object.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_OPENCHANNEL
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8 + ulBoardNameSize
Data		
ulBoardNameSize	DWORD	Length of the board name
szBoardname	CHAR[ulBoardNameSize]	Board name to open
ulChannel	DWORD	Channel Number (0..n)

Table 44: MARSHALLER_DRV_METHODID_OPENCHANNEL - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
hChannel	DWORD	Handle of the communication channel. Needed for channel related functions xChannel.....

Table 45: MARSHALLER_DRV_METHODID_OPENCHANNEL - Result

6.2.8 MARSHALLER_DRV_METHODID_OPENSYSDEV

Corresponding cifX API function: *xSysdeviceOpen()*

The returned handle needs to be used inside the netXMarshaller header to access the created system device object.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Driver object handle (created through the class factory)
ulMethodId	DWORD	MARSHALLER_DRV_METHODID_OPENSYSDEV
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4 + *ulBoardNameSize*
Data		
ulBoardNameSize	DWORD	Length of the board name
szBoardname	CHAR[ulBoardNameSize]	Board name to open

Table 46: MARSHALLER_DRV_METHODID_OPENSYSDEV - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
hSysdevice	DWORD	Handle of the system device (system channel). Needed to call system device related functions xSystemdevice....

Table 47: MARSHALLER_DRV_METHODID_OPENSYSDEV - Result

6.3 System Device Object - MethodID

The system device object is only accessible if the user has prior acquired a handle to a system device via the driver object by calling *MARSHALLER_DRV_METHODID_OPENSYSDEV*. The returned handle from is needed for all methods (calls) to the system device functions described in this chapter.

6.3.1 MARSHALLER_SYSDEV_METHODID_CLOSE

Corresponding cifX API function: *xSysdeviceClose()*

ATTENTION: This call will invalidate the system device object. The handle must not be used in any further system device functions. To re-access the system device a new call die driver object with *MARSHALLER_DRV_METHODID_OPENSYSDEV* needs to be performed.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_CLOSE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 48: MARSHALLER_SYSDEV_METHODID_CLOSE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 49: MARSHALLER_SYSDEV_METHODID_CLOSE - Result

6.3.2 MARSHALLER_SYSDEV_METHODID_INFO

Corresponding cifX API function: *xSysdeviceInformation()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_INFO
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8
Data		
ulCmd	DWORD	Command to execute (For details, see reference [2])
ulSize	DWORD	Size of the returned structure

Table 50: MARSHALLER_SYSDEV_METHODID_INFO - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulSize* from Request
Data		
abData	BYTE[ulSize]	Returned data from function call

Table 51: MARSHALLER_SYSDEV_METHODID_INFO - Result

6.3.3 MARSHALLER_SYSDEV_METHODID_RESET

Corresponding cifX API function: *xSysdeviceReset()*

ATTENTION: If the server is running on the target (e.g. netIC) the request will time out. This will also invalidate all acquired handles and the connection must be re-established. Sending requests with old handles may crash the firmware.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_RESET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulTimeout	DWORD	Timeout in ms to wait for reset to complete

Table 52: MARSHALLER_SYSDEV_METHODID_RESET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 53: MARSHALLER_SYSDEV_METHODID_RESET - Result

6.3.4 MARSHALLER_SYSDEV_METHODID_GETMBXSTATE

Corresponding cifX API function: *xSysdeviceGetMBXState()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_GETMBXSTATE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 54: MARSHALLER_SYSDEV_METHODID_GETMBXSTATE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	8
Data		
ulRecvPktCount	DWORD	Number of packets waiting on the target to be received
ulSendPktCount	DWORD	Number of packets, that can be send to the device

Table 55: MARSHALLER_SYSDEV_METHODID_GETMBXSTATE - Result

6.3.5 MARSHALLER_SYSDEV_METHODID_PUTPACKET

Corresponding cifX API function: xSysdevicePutPacket*()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_PUTPACKET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8 + *ulSendSize*
Data		
ulSendSize	DWORD	Length of following message
abData	BYTE[ulSendSize]	Packet data to send
ulTimeout	DWORD	Timeout in ms for function call

Table 56: MARSHALLER_SYSDEV_METHODID_PUTPACKET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 57: MARSHALLER_SYSDEV_METHODID_PUTPACKET - Result

6.3.6 MARSHALLER_SYSDEV_METHODID_GETPACKET

Corresponding cifX API function: xSysdeviceGetPacket*()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_GETPACKET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8 + ulSendSize
Data		
ulSize	DWORD	Maximum length of receive buffer
ulTimeout	DWORD	Timeout in ms for function call

Table 58: MARSHALLER_SYSDEV_METHODID_GETPACKET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	Total length of received rcX packet
Data		
abData	BYTE[ulSize]	Received rcX Packet

Table 59: MARSHALLER_SYSDEV_METHODID_GETPACKET - Result

6.3.7 MARSHALLER_SYSDEV_METHODID_DOWNLOAD

Corresponding cifX API function: *xSysdeviceDownload()*

The *xSysdeviceDownload*() function from the cifX API expects to transfers whole files in one transfer.

This assumes the whole file data can be stored in RAM before writing it to the file storage. But this is not possible on some target systems because of the small amount of system memory.

To enable a download for such small systems, the download in the netXMarshaller is handled by an rcX packet based download. This packet based download is processed via *xSysdevicePutPacket()* and *xSysdeviceGetPacket()* functions and because of this, no netXMarshaller data packets are currently defined.

6.3.8 MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE

Corresponding cifX API function: *xSysdeviceFindFirstFile()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	37
Data		
ulChannel	DWORD	Channel number (0..n)
hList	DWORD	0
szFileName	BYTE[16]	File name / search string / wild card search string
bFiletype	BYTE	0
ulFilesize	DWORD	0
ulRecvPktCallback	DWORD	Callback handle always 0
ulUser	DWORD	User data always 0

Table 60: MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX error code
ulSize	DWORD	25
Data		
hList	DWORD	Handle from file search function
szFileName	BYTE[16]	File name as 0 terminated string
bFiletype	BYTE	File type
ulFilesize	DWORD	File size in bytes

Table 61: MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE - Result

6.3.9 MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE

Corresponding cifX API function: *xSysdeviceFindNextFile ()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	37
Data		
ulChannel	DWORD	Channel number (0..n)
hList	DWORD	Handle from call to MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE or from previous call to MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE
szFileName	BYTE[16]	File name / search string / wild card search string
bFiletype	BYTE	0
ulFilesize	DWORD	0
ulRecvPktCallback	DWORD	Callback handle always 0
ulUser	DWORD	User data always 0

Table 62: MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX error code
ulSize	DWORD	25
Data		
hList	DWORD	Handle from file search function
szFileName	BYTE[16]	File name as 0 terminated string
bFiletype	BYTE	File type
ulFilesize	DWORD	File size in bytes

Table 63: MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE - Result

6.3.10 MARSHALLER_SYSDEV_METHODID_UPLOAD

Corresponding cifX API function: *xSysdeviceUpload()*

The *xSysdeviceUpload()* function from the cifX API expects to receive whole files in one transfer.

This assumes the whole file can read from the target file storage and sent in one transfer to the host system. But this is not possible on some target systems because of the small amount of system memory to buffer the file data.

To enable an upload for such small systems, the upload in the netXMarshaller is handled by an rcX packet based upload. This packet based upload is processed via *xSysdevicePutPacket()* and *xSysdeviceGetPacket()* functions and because of this, no netXMarshaller data packets are currently defined.

6.3.11 MARSHALLER_SYSDEV_METHODID_RESETEX

Corresponding cifX API function: *xSysdeviceResetEx()*

ATTENTION: If the server is running on the target (e.g. netIC) the request will time out. This will also invalidate all acquired handles and the connection must be re-established. Sending requests with old handles may crash the firmware.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_SYSDEV_METHODID_RESETEX
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8
Data		
ulTimeout	DWORD	Timeout in ms to wait for reset to complete
ulMode	DWORD	Reset mode and parameter

Table 64: MARSHALLER_SYSDEV_METHODID_RESETEX - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 65: MARSHALLER_SYSDEV_METHODID_RESETEX - Result

6.4 Channel Object - MethodID

The communication channel object is only accessible if the user has acquired a handle to a channel via a driver object, by calling MARSHALLER_DRV_METHODID_OPENCHANNEL. The returned handle is valid for all methods described in this chapter.

6.4.1 MARSHALLER_CHANNEL_METHODID_CLOSE

Corresponding cifX API function: *xChannelClose()*

ATTENTION: This call will invalidate the communication channel object. The handle must not be used in any further packet. To re-access the channel, a new call to MARSHALLER_DRV_METHODID_OPENCHANNEL needs to be performed on the driver object.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle acquired from MARSHALLER_DRV_METHODID_OPENCHANNEL
ulMethodId	DWORD	1
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 66: MARSHALLER_CHANNEL_METHODID_CLOSE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 67: MARSHALLER_CHANNEL_METHODID_CLOSE - Result

6.4.2 MARSHALLER_CHANNEL_METHODID_DOWNLOAD

Corresponding cifX API function: *xChannelDownload()*

The *xChannelDownload()* function from the cifX API expects to transfers whole files in one transfer.

This assumes the whole file data can be stored in RAM before writing it to the file storage. But this is not possible on some target systems because of the small amount of system memory.

To enable a download for such small systems, the download in the netXMarshaller is handled by an rcX packet based download. This packet based download is processed via *xChannelPutPacket()* and *xChannelGetPacket()* functions and because of this, no netXMarshaller data packets are currently defined.

6.4.3 MARSHALLER_CHANNEL_METHODID_GETMBXSTATE

Corresponding cifX API function: *xChannelGetMBXState()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_GETMBXSTATE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 68: MARSHALLER_CHANNEL_METHODID_GETMBXSTATE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	8
Data		
ulRecvPktCount	DWORD	Number of pending packets on device
ulSendPktCount	DWORD	Number of send able packets on device

Table 69: MARSHALLER_CHANNEL_METHODID_GETMBXSTATE - Result

6.4.4 MARSHALLER_CHANNEL_METHODID_PUTPACKET

Corresponding cifX API function: *xChannelPutPacket()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_PUTPACKET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8 + ulSendSize
Data		
ulSendSize	DWORD	Length of following message
abData	BYTE[ulSendSize]	Packet data to send
ulTimeout	DWORD	Timeout in [ms]

Table 70: MARSHALLER_CHANNEL_METHODID_PUTPACKET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 71: MARSHALLER_CHANNEL_METHODID_PUTPACKET - Result

6.4.5 MARSHALLER_CHANNEL_METHODID_GETPACKET

Corresponding cifX API function: *xChannelGetPacket()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_GETPACKET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8
Data		
ulSendSize	DWORD	Length of following message
ulTimeout	DWORD	Timeout in [ms]

Table 72: MARSHALLER_CHANNEL_METHODID_GETPACKET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulSendSize* from Request
Data		
abData	BYTE[Size]	Returned packet

Table 73: MARSHALLER_CHANNEL_METHODID_GETPACKET – Result

6.4.6 MARSHALLER_CHANNEL_METHODID_GETSENDPACKET

Corresponding cifX API function: *xChannelGetSendPacket()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_GETSENDPACKET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulSendSize	DWORD	Length of following message

Table 74: MARSHALLER_CHANNEL_METHODID_GETSENDPACKET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulSendSize* from Request
Data		
abData	BYTE[ulSize]	Returned packet

Table 75: MARSHALLER_CHANNEL_METHODID_GETSENDPACKET - Result

6.4.7 MARSHALLER_CHANNEL_METHODID_CONFIGLOCK

Corresponding cifX API function: *xChannelConfigLock()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_CONFIGLOCK
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Command to execute
ulState	DWORD	State to set
ulTimeout	DWORD	Timeout in ms for function call

Table 76: MARSHALLER_CHANNEL_METHODID_CONFIGLOCK - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
ulState	DWORD	Returned State

Table 77: MARSHALLER_CHANNEL_METHODID_CONFIGLOCK - Result

6.4.8 MARSHALLER_CHANNEL_METHODID_RESET

Corresponding cifX API function: *xChannelReset()*

ATTENTION: If the server is running on the target (e.g. netIC) a '*SYSTEM_START*' request will time out. This will also invalidate all acquired handles and the connection must be re-established. Sending requests with old handles may crash the firmware.

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_RESET
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8
Data		
ulMode	DWORD	Reset mode
ulTimeout	DWORD	Timeout in [ms] for function call

Table 78: MARSHALLER_CHANNEL_METHODID_RESET - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 79: MARSHALLER_CHANNEL_METHODID_RESET - Result

6.4.9 MARSHALLER_CHANNEL_METHODID_INFO

Corresponding cifX API function: *xChannelInfo()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_INFO
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulSize	DWORD	Requested Size

Table 80: MARSHALLER_CHANNEL_METHODID_INFO - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulSize* from Request
Data		
abData	BYTE[ulSize]	Returned Information

Table 81: MARSHALLER_CHANNEL_METHODID_INFO - Result

6.4.10 MARSHALLER_CHANNEL_METHODID_WATCHDOG

Corresponding cifX API function: *xChannelWatchdog()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_WATCHDOG
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	8
Data		
ulCmd	DWORD	Watchdog command
ulTrigger	DWORD	Trigger Value

Table 82: MARSHALLER_CHANNEL_METHODID_WATCHDOG - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
ulTrigger	DWORD	Returned trigger value

Table 83: MARSHALLER_CHANNEL_METHODID_WATCHDOG - Result

6.4.11 MARSHALLER_CHANNEL_METHODID_HOSTSTATE

Corresponding cifX API function: *xChannelHostState()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_HOSTSTATE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Host State command
ulState	DWORD	State to set
ulTimeout	DWORD	Timeout in [ms]

Table 84: MARSHALLER_CHANNEL_METHODID_HOSTSTATE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
ulState	DWORD	Returned state

Table 85: MARSHALLER_CHANNEL_METHODID_HOSTSTATE - Result

6.4.12 MARSHALLER_CHANNEL_METHODID_IOREAD

Corresponding cifX API function: *xChannelIORead()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_IOREAD
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	16
Data		
ulAreaNumber	DWORD	Input area number (0..n)
ulOffset	DWORD	Offset in input area (0..n)
ulTimeout	DWORD	Timeout in [ms]
ulDataLen	DWORD	Length of requested data

Table 86: MARSHALLER_CHANNEL_METHODID_IOREAD - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulDataLen* from request or maximum buffer size of the target if this is smaller than the requested length
Data		
abData	BYTE[ulSize]	Returned input data

Table 87: MARSHALLER_CHANNEL_METHODID_IOREAD - Result

6.4.13 MARSHALLER_CHANNEL_METHODID_IOWRITE

Corresponding cifX API function: *xChannelIOWrite()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_IOWRITE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	16 + *ulDataLen*
Data		
ulAreaNumber	DWORD	Output area number (0..n)
ulOffset	DWORD	Offset in output area (0..n)
ulTimeout	DWORD	Timeout in [ms}
ulDataLen	DWORD	Length of output data in *abData*
abData	BYTE[ulDataLen]	Data to write

Table 88: MARSHALLER_CHANNEL_METHODID_IOWRITE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 89: MARSHALLER_CHANNEL_METHODID_IOWRITE - Result

6.4.14 MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA

Corresponding cifX API function: *xChannelIOReadSendData()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulAreaNumber	DWORD	Output area number (0..n)
ulOffset	DWORD	Offset in output area (0..n)
ulDataLen	DWORD	Length of requested data

Table 90: MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulDataLen* from request or maximum buffer size of the target if this is smaller than the requested length
Data		
abData	BYTE[ulSize]	Return data from the output area

Table 91: MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA - Result

6.4.15 MARSHALLER_CHANNEL_METHODID_BUSSTATE

Corresponding cifX API function: *xChannelBusState()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_BUSSTATE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Command code (see reference [2])
ulState	DWORD	Unused (set to 0)
ulTimeout	DWORD	Timeout in [ms]

Table 92: MARSHALLER_CHANNEL_METHODID_BUSSTATE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
ulState	DWORD	Returned actual state

Table 93: MARSHALLER_CHANNEL_METHODID_BUSSTATE - Result

6.4.16 MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK

Corresponding cifX API function: *xChannelControlBlock()*

Request: CIFX_CMD_READ_DATA Command (see reference [2])

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Command code = CIFX_CMD_READ_DATA
ulOffset	DWORD	Offset in control block
ulDatalen	DWORD	Length of data to read

Table 94: MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Request (READ_DATA)

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulDataLen* from request
Data		
abData	BYTE[ulSize]	Returned data

Table 95: MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Result (READ_DATA)

Request: CIFX_CMD_WRITE_DATA Command (see reference [2])

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12 + *ulDataLen*
Data		
ulCmd	DWORD	Command code = CIFX_CMD_WRITE_DATA
ulOffset	DWORD	Offset in control block
ulDatalen	DWORD	Length to write
abData	BYTE[ulDataLen]	Data to write

Table 96: MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Request (WRITE_DATA)

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 97: MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Result (WRITE _DATA)

6.4.17 MARSHALLER_CHANNEL_METHODID_STATUSBLOCK

Corresponding cifX API function: *xChannelStatusBlock()*

Request: CIFX_CMD_READ_DATA Command (see reference [2])

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_STATUSBLOCK
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Command code = CIFX_CMD_READ_DATA
ulOffset	DWORD	Offset in status block
ulDatalen	DWORD	Length to read

Table 98: MARSHALLER_CHANNEL_METHODID_STATUSBLOCK - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulDataLen* from request
Data		
abData	BYTE[ulSize]	Returned data

Table 99: MARSHALLER_CHANNEL_METHODID_STATUSBLOCK - Result

6.4.18 MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK

Corresponding cifX API function: *xChannelExtendedStatusBlock()*

Request: CIFX_CMD_READ_DATA Command (see reference [2])

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Command code = CIFX_CMD_READ_DATA
ulOffset	DWORD	Offset in the extended status block
ulDataLen	DWORD	Length to read

Table 100: MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	*ulDataLen* from request
Data		
abData	BYTE[ulSize]	Returned data

Table 101: MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK - Result

6.4.19 MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE

Corresponding cifX API function: *xChannelFindFirstFile()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	37
Data		
ulChannel	DWORD	Channel number (0..n)
hList	DWORD	0
szFileName	BYTE[16]	File name / search string / wild card search string
bFiletype	BYTE	0
ulFilesize	DWORD	0
ulRecvPktCallback	DWORD	Callback handle always 0
ulUser	DWORD	User data always 0

Table 102: MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX error code
ulSize	DWORD	25
Data		
hList	DWORD	Handle from file search function
szFileName	BYTE[16]	File name as 0 terminated string
bFiletype	BYTE	File type
ulFilesize	DWORD	File size in bytes

Table 103: MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE - Result

6.4.20 MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE

Corresponding cifX API function: *xChannelFindNextFile()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the communication channel
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	37
Data		
ulChannel	DWORD	Channel number (0..n)
hList	DWORD	Handle from call to MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE or from previous call to MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE
szFileName	BYTE[16]	File name / search string / wild card search string
bFiletype	BYTE	0
ulFilesize	DWORD	0
ulRecvPktCallback	DWORD	Callback handle always 0
ulUser	DWORD	User data always 0

Table 104: MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX error code
ulSize	DWORD	25
Data		
hList	DWORD	Handle from file search function
szFileName	BYTE[16]	File name as 0 terminated string
bFiletype	BYTE	File type
ulFilesize	DWORD	File size in bytes

Table 105: MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE - Result

6.4.21 MARSHALLER_CHANNEL_METHODID_UPLOAD

Corresponding cifX API function: *xChannelUpload()*

The *xChannelUpload()* function from the cifX API expects to receive whole files in one transfer.

This assumes the whole file can read from the target file storage and sent in one transfer to the host system. But this is not possible on some target systems because of the small amount of system memory to buffer the file data.

To enable an upload for such small systems, the upload in the netXMarshaller is handled by an rcX packet based upload. This packet based upload is processed via *xChannelPutPacket()* and *xChannelGetPacket()* functions and because of this, no netXMarshaller data packets are currently defined.

6.4.22 MARSHALLER_CHANNEL_METHODID_IOINFO

Corresponding cifX API function: *xChannelIOInfo()*

Request

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	Handle of the system device
ulMethodId	DWORD	MARSHALLER_CHANNEL_METHODID_IOINFO
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	12
Data		
ulCmd	DWORD	Command CIFX_IO_INPUT_AREA CIFX_IO_OUTPUT_AREA
ulArea	DWORD	Area number (0..1)
ulDataLen	DWORD	sizeof(CHANNEL_IO_INFORMATION)

Table 106: MARSHALLER_CHANNEL_METHODID_IOINFO - Request

Result

Element	Data type	Value
Marshaller Header		
ulHandle	DWORD	*ulHandle* from request
ulMethodId	DWORD	*ulMethodId* from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX error code
ulSize	DWORD	sizeof(CHANNEL_IO_INFORMATION)
Data		
tChannelInfo	CHANNEL_IO_INFORMATION	Channel IO information structure

Table 107: MARSHALLER_CHANNEL_METHODID_IOINFO – Result

6.5 Creating a connection and calling functions

Before using any function of the cifX API it is necessary to initialize the netXMarshaller component on the host and the target.

The following sequence is mandatory to be able to communicate with a device through the '*netX Diagnostic and Remote Access*' services.

Note: These steps are for the user, as they are "hidden" in the cifX function wrappers of the netXMarshaller.

To establish a new connection the following steps need to be done:

- Send a request to the default '*Class Factory Object*' to create a class factory for the application:

Request data		**Returned data**	
ulHandle	0	ulHandle	0
ulMethodID	1 MARSHALLER_CF _METHODID_CREATEINSTANCE	ulMethodID	1 MARSHALLER_CF _METHODID_CREATEINSTANCE
ulSize	4	ulSize	4
ulData	0 (ClassFactoryObjectID)	ulData	***Class Factory Handle***

- Create a driver object:

Request data		**Returned data**	
ulHandle	***Class Factory Handle***	ulHandle	***Class Factory Handle***
ulMethodID	1 MARSHALLER_CF _METHODID_CREATEINSTANCE	ulMethodID	1 MARSHALLER_CF _METHODID_CREATEINSTANCE
ulSize	4	ulSize	4
ulData	1 (DriverObjectID)	ulData	***Driver Handle***

- Call *xDriverOpen()* on the driver object

Request data		**Returned data**	
ulHandle	***Driver Handle***	ulHandle	***Driver Handle***
ulMethodID	1 MARSHALLER_DRV _METHODID_OPEN	ulMethodID	1 MARSHALLER_DRV _METHODID_OPEN
ulSize	0	ulSize	4

6.5.1 Enumerating boards

The enumeration needs the '*Driver Handle*' retrieved during the creation of a connection (see above). The following sequence shows enumeration of devices on a target.

It only shows the enumeration for board number 0 and channel number 0. To enumerate all devices the board number (*ulBoard*) and channel number (*ulChannel*) must be incremented until the error *CIFX_NO_MORE_ENTRIES* is returned in the answer.

- Call *xDriverEnumBoards()* on the driver object

Request data		**Returned data**	
ulHandle	***Driver Handle***	ulHandle	***Driver Handle***
ulMethodID	5 MARSHALLER_DRV _METHODID_ENUMBOARDS	ulMethodID	5 MARSHALLER_DRV _METHODID_ENUMBOARDS
ulSize	8	ulSize	sizeof(BOARD_INFORMATION)
ulBoardNr	0	ulError	CIFX_NO_ERROR or CIFX_NO_MORE_ENTRIES, if enumeration is finished
ulDataSize	sizeof(BOARD_INFORMATION)	abData	BOARD_INFORMATION

- Call *xDriverEnumChannels()* on the driver object

Request data		**Returned data**	
ulHandle	***Driver Handle***	ulHandle	***Driver Handle***
ulMethodID	6 MARSHALLER_DRV _METHODID_ENUMCHANNELS	ulMethodID	6 MARSHALLER_DRV _METHODID_ENUMCHANNELS
ulSize	12	ulSize	sizeof(CHANNEL_INFORMATION
ulBoardNr	0	ulError	CIFX_NO_ERROR or CIFX_NO_MORE_ENTRIES, if enumeration is finished
ulChannel	0	abData	CHANNEL_INFORMATION
ulDataSize	sizeof(CHANNEL_INFORMATION)		

6.5.2 Opening a System Device

To access a system device using the marshaller, a new object needs to be created on the target. This is done by sending *MARSHALLER_DRV_METHODID_OPENSYSDEV* to the driver object. The returned handle needs to be used on all further calls to the system device.

- Call *xSysdeviceOpen()* on the driver object

Request data		**Returned data**	
ulHandle	***Driver Handle***	ulHandle	***Driver Handle***
ulMethodID	9 MARSHALLER_DRV _METHODID_ENUMCHANNELS	ulMethodID	9 MARSHALLER_DRV _METHODID_ENUMCHANNELS
ulSize	9	ulSize	4
ulBoardNameSize	5	ulData	***Sysdevice Handle***
szDeviceName	"cifX0"		

- Call *xSysdeviceGetMBXState()* on the system device object

Request data		**Returned data**	
ulHandle	***Sysdevice Handle***	ulHandle	***Driver Handle***
ulMethodID	4 MARSHALLER_SYSDEV _METHODID_GETMBXSTATE	ulMethodID	4 MARSHALLER_SYSDEV _METHODID_GETMBXSTATE
ulSize	0	ulSize	8
		ulRecvPktCount	x
		ulSendPktCount	y

- Close the system device by calling *xSysdeviceClose()* on the system device object

  Note: This will invalidate the system device handle

Request data		**Returned data**	
ulHandle	***Sysdevice Handle***	ulHandle	***Driver Handle***
ulMethodID	1 MARSHALLER_SYSDEV _METHODID_CLOSE	ulMethodID	1 MARSHALLER_SYSDEV _METHODID_CLOSE
ulSize	0	ulSize	0

6.5.3 Opening a Communication Channel

For calling functions on a communications channel, a new object needs to be created on the target. This is done by sending *MARSHALLER_DRV_METHODID_OPENCHANNEL* to the driver object. The returned handle needs to be used on all further calls to the communication channel.

- Call *xChannelOpen()* on the driver object

Request data		**Returned data**	
ulHandle	***Driver Handle***	ulHandle	***Driver Handle***
ulMethodID	8 MARSHALLER_DRV _METHODID_OPENCHANNEL	ulMethodID	8 MARSHALLER_DRV _METHODID_OPENCHANNEL
ulSize	13	ulSize	4
ulBoardName	5	ulData	***Channel Handle***
DeviceName	"cifX0"		

- Call xChannelGetMBXState on the system device object

Request data		**Returned data**	
ulHandle	***Channel Handle***	ulHandle	***Channel Handle***
ulMethodID	3 MARSHALLER_CHANNEL _METHODID_GETMBXSTATE	ulMethodID	3 MARSHALLER_CHANNEL _METHODID_GETMBXSTATE
ulSize	0	ulSize	8
		ulRecvPktCount	x
		ulSendPktCount	y

- Close the communication channel by calling *xChannelClose()* on the communication channel object.

  Note: This will invalidate the communication channel handle

Request data		**Returned data**	
ulHandle	***Channel Handle***	ulHandle	***Channel Handle***
ulMethodID	1 MARSHALLER_CHANNEL _METHODID_CLOSE	ulMethodID	1 MARSHALLER_CHANNEL _METHODID_CLOSE
ulSize	0	ulSize	0

6.5.4 netXMarshaller data examples

1) Create ClassFactory (Generic Marshaller Handler, CFMethodID = valid)

					Create Class Factory Cmd
0	1	1	0	4	0
SYC_HANDLE	MethodID	Status	Error	Size	Data (DWORD)

0	1	0	0	4	e.g. 0x345AD8
SYC_HANDLE	MethodID	Status	Error	Size	Handle to Class Factory

2) Query Server Version

e.g. 0x345AD8	0	1	0	0
SYC_HANDLE	MethodID	Status	Error	Size

e.g. 0x345AD8	0	0	0	4	0x00900000
SYC_HANDLE	MethodID	Status	Error	Size	ulVersion

3) Create Driver Object (for DMethodID methods)

					Create Driver
e.g. 0x345AD8	0	1	0	4	1
SYC_HANDLE	MethodID	Status	Error	Size	Data (DWORD)

e.g. 0x345AD8	0	0	0	4	e.g. 0x345AA0
SYC_HANDLE	MethodID	Status	Error	Size	Handle to Driver

4) Call xDriverOpen

e.g. 0x345AA0	1	1	0	0
SYC_HANDLE	MethodID	Status	Error	Size

e.g. 0x345AA0	0	0	0	0
SYC_HANDLE	MethodID	Status	Error	Size

5) Call xDriverEnumBoards (Board0)

e.g. 0x345AA0	5	1	0	8	0	0x6D
SYC_HANDLE	MethodID	Status	Error	Size	ulBoard	ulSize

e.g. 0x345AA0	5	0	0	0x6D	...
SYC_HANDLE	MethodID	Status	Error	Size	BOARD_INFORMATION structure

6) Call xDriverEnumChannels (Board0, Channel0)

e.g. 0x345AA0	6	1	0	0x0C	0	0	0xA4
SYC_HANDLE	MethodID	Status	Error	Size	ulBoard	ulChannel	ulSize

e.g. 0x345AA0	6	0	0	0xA4	...
SYC_HANDLE	MethodID	Status	Error	Size	CHANNEL_INFORMATION structure

7) Call xDriverEnumChannels (Board0, Channel1)

e.g. 0x345AA0	6	1	0	0x0C	0	1	0xA4
SYC_HANDLE	MethodID	Status	Error	Size	ulBoard	ulChannel	ulSize

e.g. 0x345AA0	6	0	0x800A014	0
SYC_HANDLE	MethodID	Status	Error	Size

8) Call xDriverEnumBoards (Board1)

e.g. 0x345AA0	5	1	0	8	1	0x6D
SYC_HANDLE	MethodID	Status	Error	Size	ulBoard	ulSize

e.g. 0x345AA0	5	0	0x800A014	0
SYC_HANDLE	MethodID	Status	Error	Size

9) Call xChannelOpen („cifX0", Channel0)

e.g. 0x345AA0	8	1	0	13	5	„cifX0"	0
SYC_HANDLE	MethodID	Status	Error	Size	Boradnamesize	szBoardname	ulChannel

e.g. 0x345AA0	8	0	0	4	e.g. 0x360A50
SYC_HANDLE	MethodID	Status	Error	Size	Handle to Channel

10) Call xChannelIOWrite

e.g. 0x360A50	0x13	1	0	20	0	0	0	4	...
SYC_HANDLE	MethodID	Status	Error	Size	ulArea	ulOffset	ulTimeout	ulDataLen	Data

e.g. 0x360A50	0x13	0	0	0
SYC_HANDLE	MethodID	Status	Error	Size

11) Call xChannelIOWrite

e.g. 0x360A50	0x12	1	0	16	0	0	0	4
SYC_HANDLE	MethodID	Status	Error	Size	ulArea	ulOffset	ulTimeout	ulDataLen

e.g. 0x360A50	0x12	0	0	4	...
SYC_HANDLE	MethodID	Status	Error	Size	Data

7 Appendix

7.1 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

7.2 List of tables

7.3 List of figures

7.4 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.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
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