



Program Reference Guide
netX Diagnostic and Remote Access
Fundamentals
V1.0.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC090703PR02EN | Revision 2 | English | 2011-11 | Released | Public

Table of Contents

1	Introduction.....	4
1.1	About this Document.....	4
1.2	List of Revisions	4
1.3	Terms, Abbreviations and Definitions	5
1.4	References	5
1.5	Legal Notes	6
1.5.1	Copyright	6
1.5.2	Important Notes.....	6
1.5.3	Exclusion of Liability	7
1.5.4	Export	7
2	Architectural Basis.....	8
2.1	Overview	8
2.2	Software Structure	9
2.3	cifX Application Interface.....	12
2.4	User Data Transfer Modes.....	12
3	Hilscher Transport Mechanism.....	13
3.1	Data Encapsulation	14
3.2	Transport Header Definition	14
3.2.1	Data Types	15
3.2.2	State Definitions	15
3.3	Keep Alive	16
3.3.1	First Request	16
3.3.2	Continuing Keep Alive	17
3.3.3	Faulty Request	18
3.3.4	Creating an Keep Alive Request	19
3.3.5	Creating an Acknowledgement.....	20
3.3.6	Keep Alive Response	20
3.3.7	Host Functionality	21
3.3.8	Target Functionality	22
4	netXMarshaller.....	23
4.1	Target Administration Commands	23
4.2	Querying Basic Server Information	24
4.3	Query Device Information	26
4.3.1	Query Number of Devices with HIL_TRANSPORT_TYPE_RCX_PACKET	27
4.4	Fallback Behaviour.....	33
4.5	List of Error Codes	34
5	rcX Packet Transfer.....	35
5.1	Addressing Example	36
6	cifX API Data Transfer.....	37
6.1	Classfactory Object - MethodID	41
6.1.1	MARSHALLER_CF_METHODID_SERVERVERSION.....	41
6.1.2	MARSHALLER_CF_METHODID_CREATEINSTANCE.....	42
6.2	Driver Object - MethodID	43
6.2.1	MARSHALLER_DRV_METHODID_OPEN	43
6.2.2	MARSHALLER_DRV_METHODID_CLOSE	44
6.2.3	MARSHALLER_DRV_METHODID_GETINFO.....	45
6.2.4	MARSHALLER_DRV_METHODID_ERRORDESCR	46
6.2.5	MARSHALLER_DRV_METHODID_ENUMBOARDS	47
6.2.6	MARSHALLER_DRV_METHODID_ENUMCHANNELS.....	48
6.2.7	MARSHALLER_DRV_METHODID_OPENCHANNEL	49
6.2.8	MARSHALLER_DRV_METHODID_OPENSYSDEV	50
6.3	System Device Object - MethodID.....	51
6.3.1	MARSHALLER_SYSDEV_METHODID_CLOSE	51
6.3.2	MARSHALLER_SYSDEV_METHODID_INFO	52
6.3.3	MARSHALLER_SYSDEV_METHODID_RESET.....	53
6.3.4	MARSHALLER_SYSDEV_METHODID_GETMBXSTATE	54
6.3.5	MARSHALLER_SYSDEV_METHODID_PUTPACKET	55
6.3.6	MARSHALLER_SYSDEV_METHODID_GETPACKET	56
6.3.7	MARSHALLER_SYSDEV_METHODID_DOWNLOAD.....	57

6.3.8	MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE	58
6.3.9	MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE	59
6.3.10	MARSHALLER_SYSDEV_METHODID_UPLOAD	60
6.4	Channel Object - MethodID	61
6.4.1	MARSHALLER_CHANNEL_METHODID_CLOSE	61
6.4.2	MARSHALLER_CHANNEL_METHODID_DOWNLOAD	62
6.4.3	MARSHALLER_CHANNEL_METHODID_GETMBXSTATE	62
6.4.4	MARSHALLER_CHANNEL_METHODID_PUTPACKET	63
6.4.5	MARSHALLER_CHANNEL_METHODID_GETPACKET	64
6.4.6	MARSHALLER_CHANNEL_METHODID_GETSENDPACKET	65
6.4.7	MARSHALLER_CHANNEL_METHODID_CONFIGLOCK	66
6.4.8	MARSHALLER_CHANNEL_METHODID_RESET	67
6.4.9	MARSHALLER_CHANNEL_METHODID_INFO	68
6.4.10	MARSHALLER_CHANNEL_METHODID_WATCHDOG	69
6.4.11	MARSHALLER_CHANNEL_METHODID_HOSTSTATE	70
6.4.12	MARSHALLER_CHANNEL_METHODID_IOREAD	71
6.4.13	MARSHALLER_CHANNEL_METHODID_IOWRITE	72
6.4.14	MARSHALLER_CHANNEL_METHODID_IOREADSENDATA	73
6.4.15	MARSHALLER_CHANNEL_METHODID_BUSSTATE	74
6.4.16	MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK	75
6.4.17	MARSHALLER_CHANNEL_METHODID_STATUSBLOCK	77
6.4.18	MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK	78
6.4.19	MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE	79
6.4.20	MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE	80
6.4.21	MARSHALLER_CHANNEL_METHODID_UPLOAD	81
6.4.22	MARSHALLER_CHANNEL_METHODID_IOINFO	82
6.5	Createing a Connection and Calling Functions	83
6.5.1	Enumerating Boards	84
6.5.2	Opening a System Device	85
6.5.3	Opening a Communication Channel	86
6.5.4	netXMarshaller Data Examples	87
7	Appendix	89
7.1	List of Tables	89
7.2	List of Figures	90
7.3	Contacts	91

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
1	2009-07-13	RM	All	Created - netX USB Specification => USB Connection
2	2010-06-02	RM	3.2.2	State definition 0x17 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 nterface based on net X
CMD	Command
comX	C ommunication 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

This document based on the following specifications and manuals:

- [1] netX Dual-Port Memory Interface for netX based Products, Rev. 7, Hilscher GmbH, December 2008
- [2] cifX Device Driver Manual, Rev. 12, Hilscher GmbH, November 2008
- [3] ODM V3 Specification, Rev. 3, 2006-10-18

Table 3: References

1.5 Legal Notes

1.5.1 Copyright

© 2009-2010 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

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

1.5.2 Important Notes

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

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

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

1.5.3 Exclusion of Liability

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

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

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

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

1.5.4 Export

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

2 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

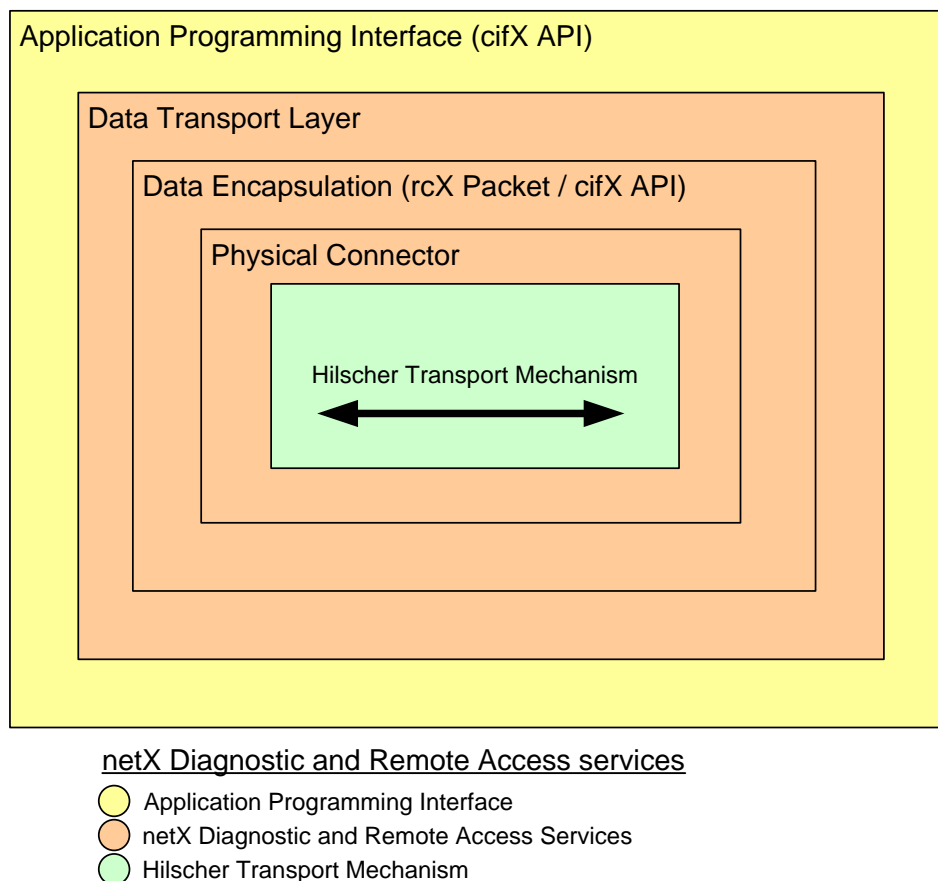


Figure 1: Overview

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.

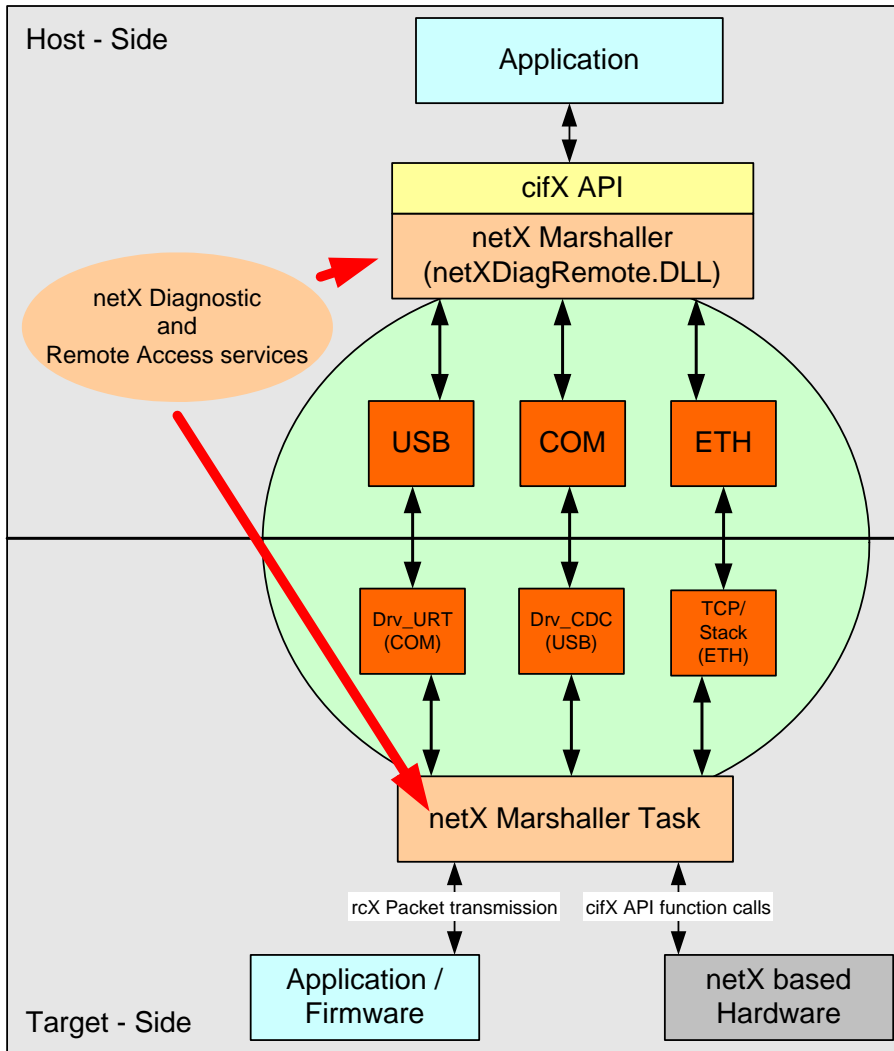


Figure 2 : Software Structure

The 'Host-Side' is created for Microsoft Windows based systems, 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 by own programming reference manuals.

Internal Block Diagram:

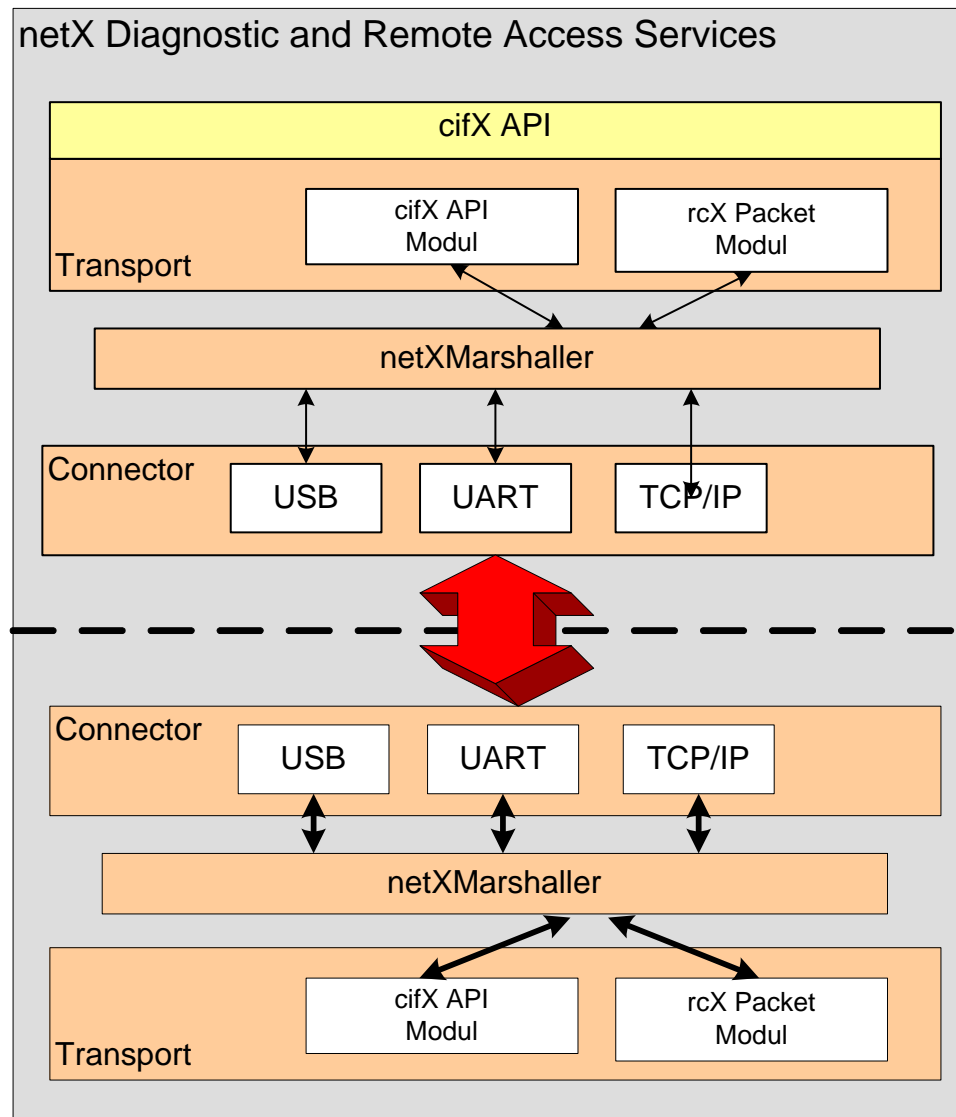


Figure 3: Internal Block Diagram

Internal Data Flow:

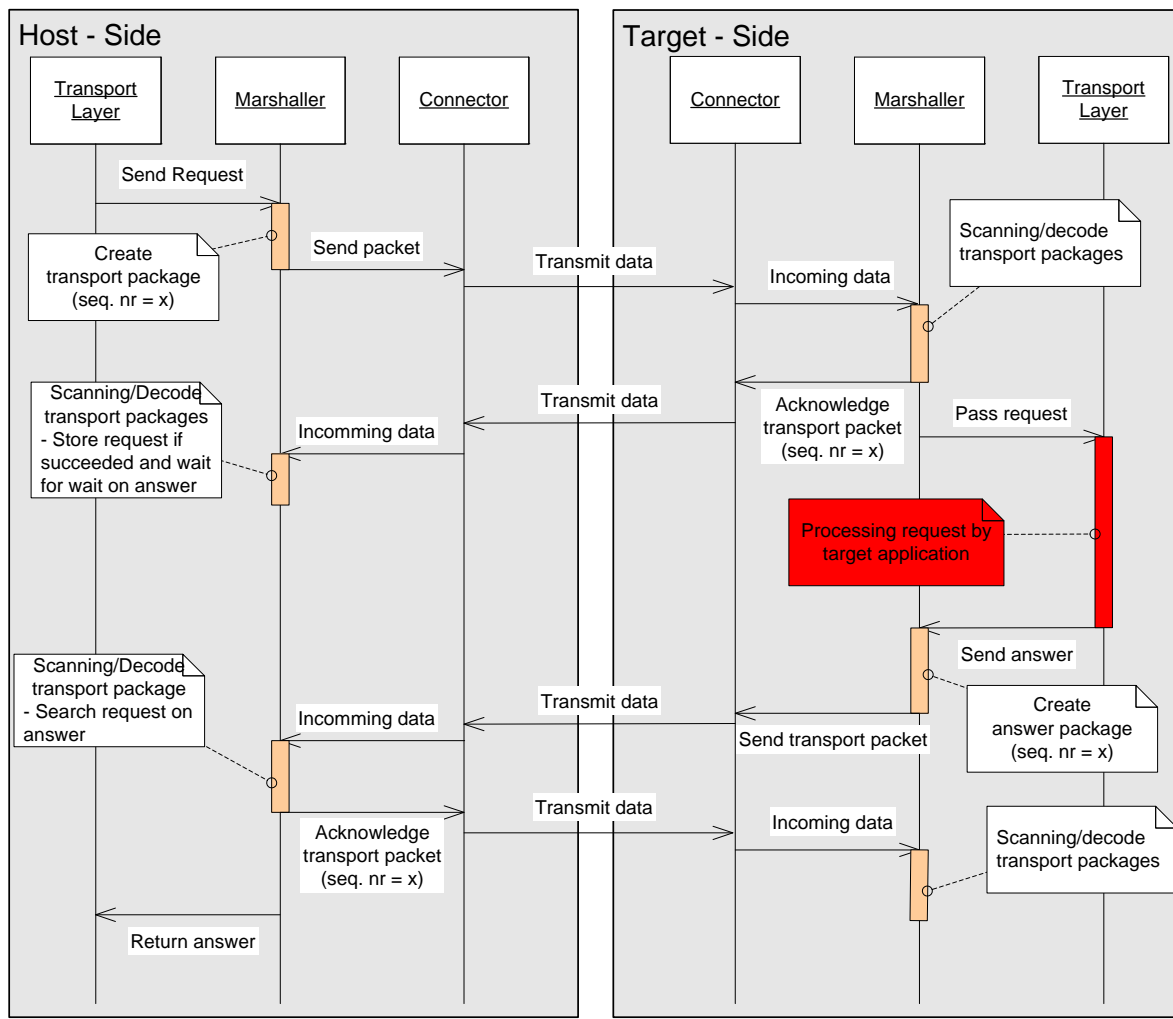


Figure 4 : Internal Data Flow

2.3 cifX Application Interface

The 'netX Diagnostic and Remote Access' services are accessible by an application via the cifX API. This application interface is defined for the '*cifX Device Driver*' and described in the '*cifX Device Driver Manual*'.

2.4 User Data Transfer Modes

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

Which kind of data transfer is used, is defined by the target system. The netXMarshaller internally handles the transfer corresponding to definition.

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.

Transfer Modes:

- rcX Packet transfer
- cifX API remote procedure calls

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 can't be used to assign answers to request.

3.1 Data Encapsulation

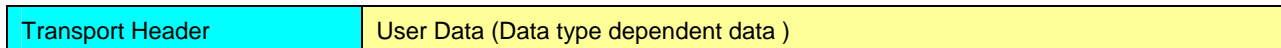


Figure 5 : Data Encapsulation

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 (<i>bDevice</i>) 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
usTransactionID	WORD	Transaction ID 0 not supported 0x0001..0x7FFF Host request package 0x8000 - 0xFFFF Device request package Note: Highest bit reserved for device request
usReserved	WORD	Reserved for future use
Package Data		
-/-	-/-	Packet data - Content defined by <i>usDataType</i> - Length in bytes defined by <i>ulLength</i>

Table 4 Transport Header Definition

3.2.1 Data Types

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

Mandatory 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 : Mandatory 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 Type

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 7 : 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

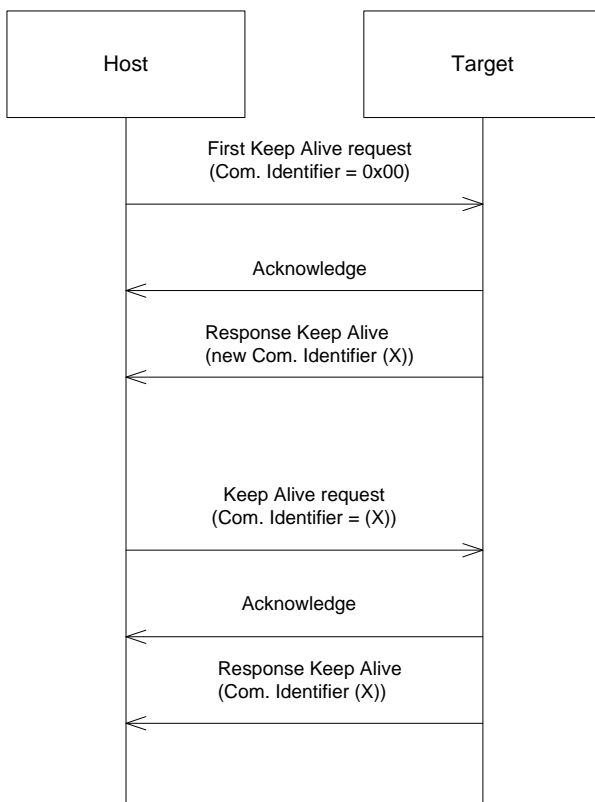


Figure 6: Keep Alive - First Request

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.

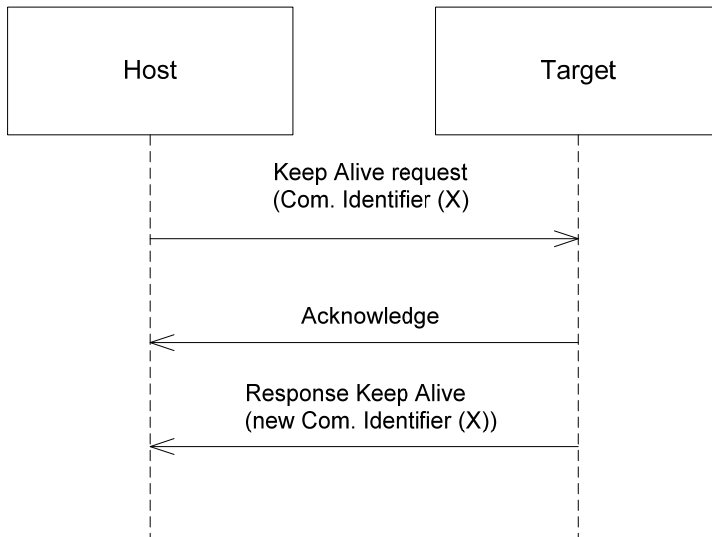


Figure 7: Keep Alive - Continuing

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.

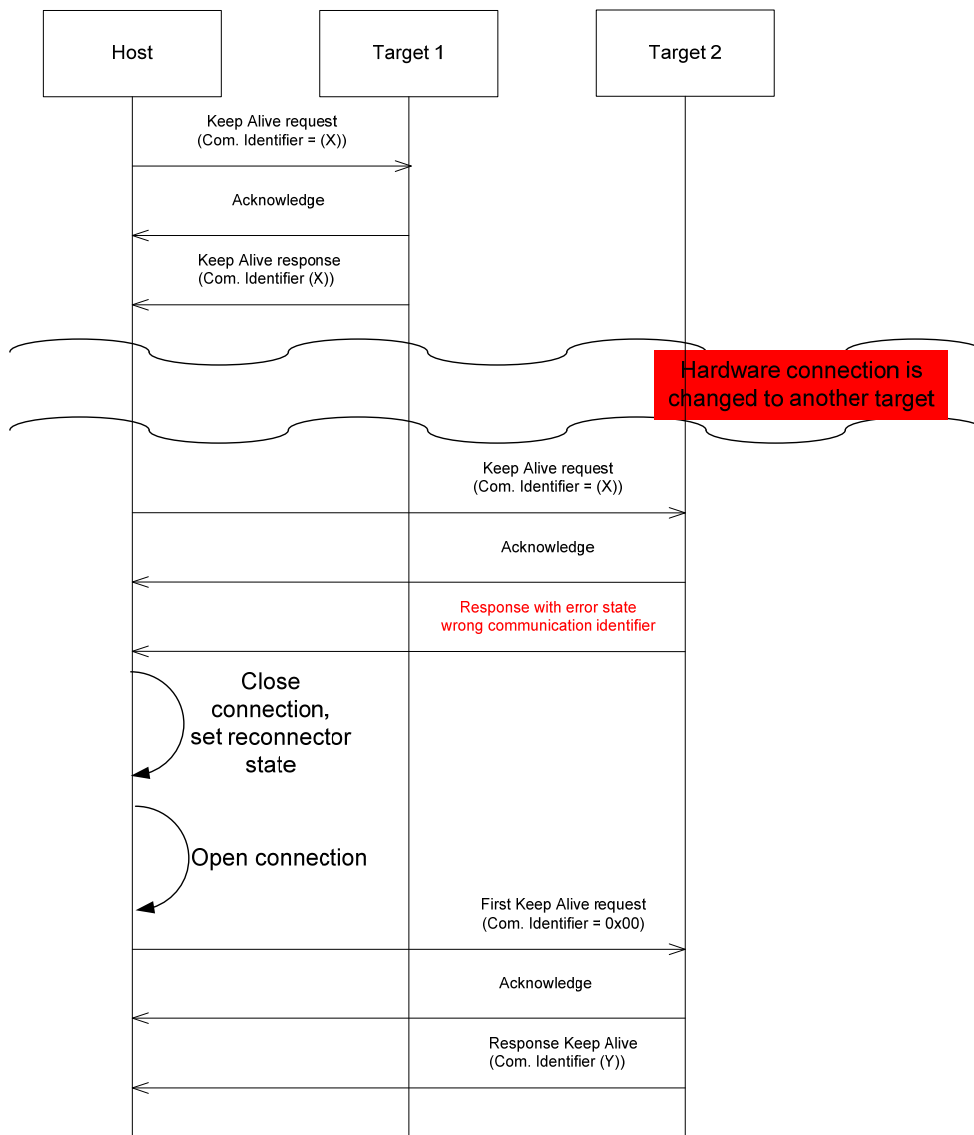


Table 8: Keep Alive - Faulty Request

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
ulReserved	UINT32	Unused/Reserved for later use
Package Data		
ulComID	UINT32	Communication identifier, unique identification of the communication partner. 0x00000000 = Request for a new identifier

Table 9: 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
ulReserved	UINT32	Unused/Reserved for later use

Table 10: 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
ulReserved	UINT32	Unused/Reserved for later use
Package Data		
ulComID	UINT32	Communication identifier, unique identification of the communication partner.

Table 11: 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, then 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.

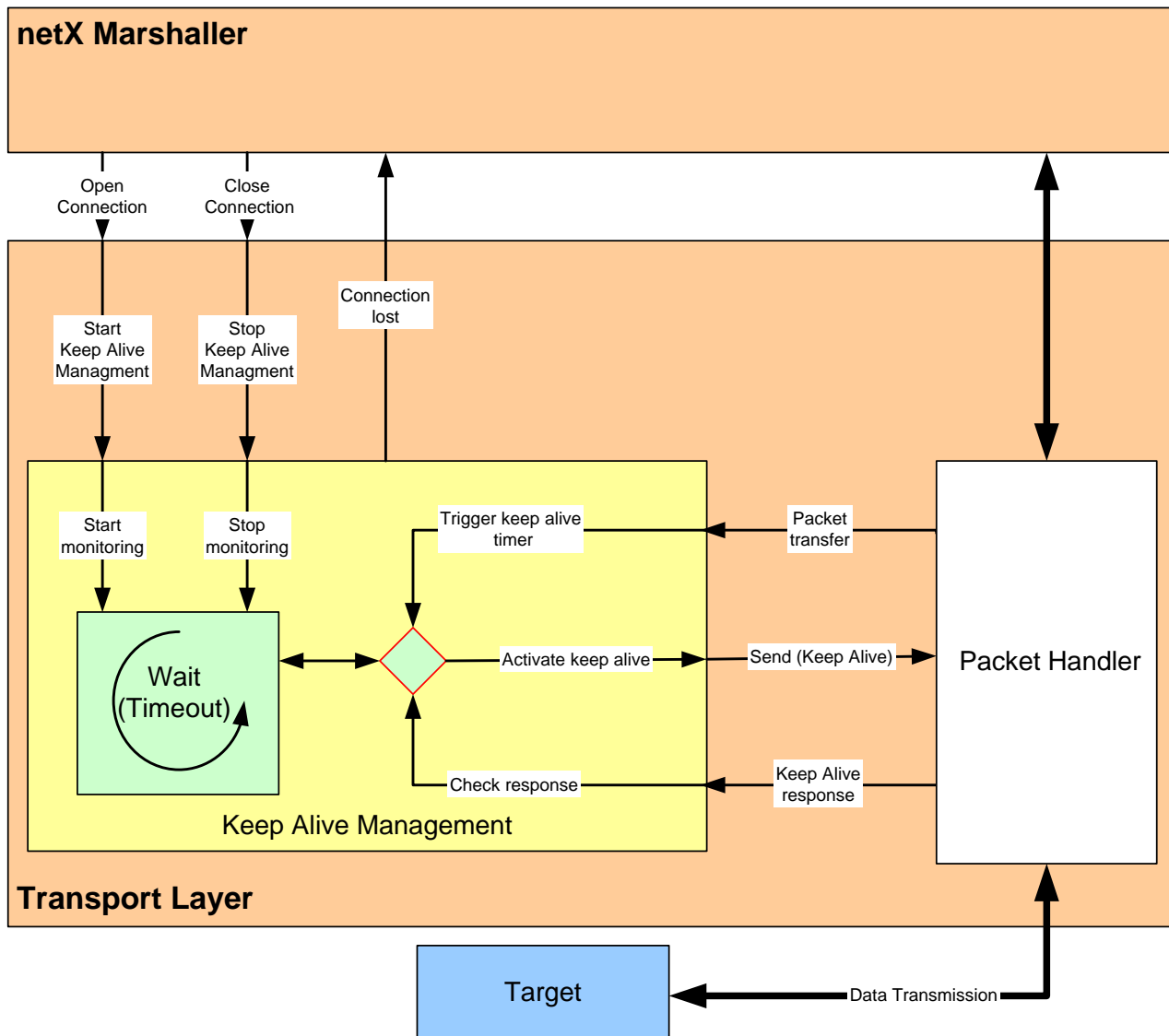


Figure 8: Keep Alive Host Functionality

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.

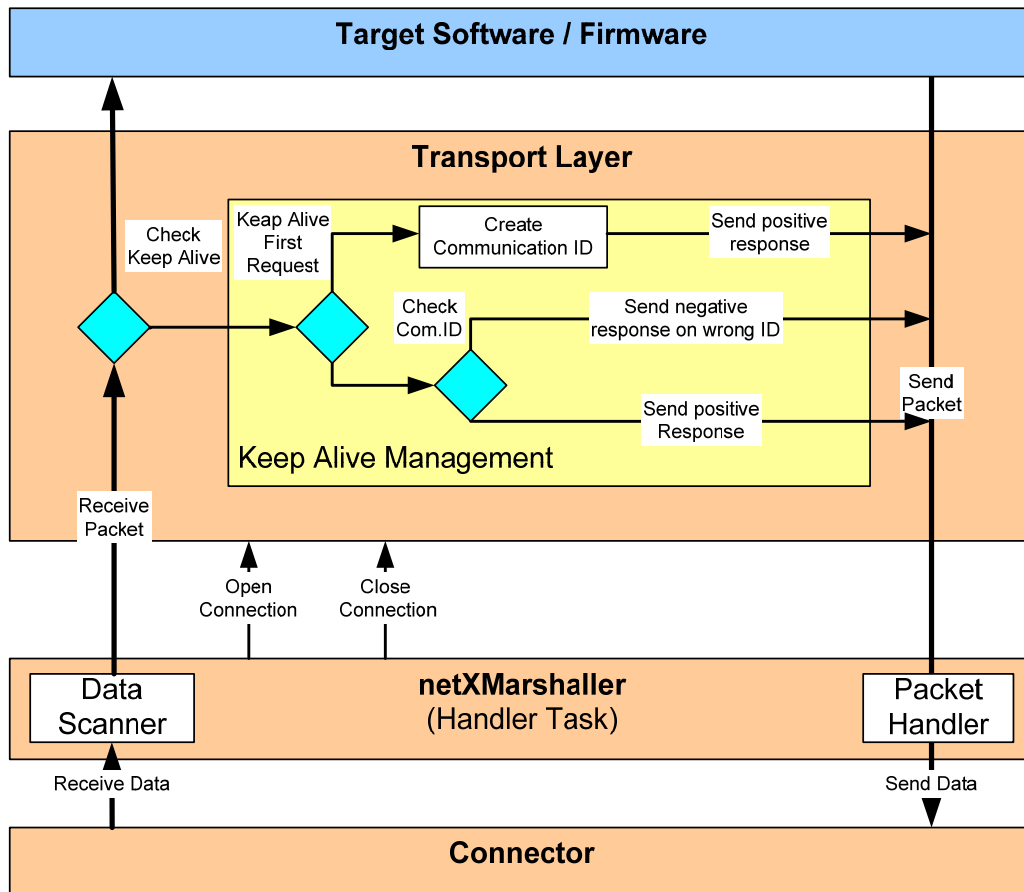


Figure 9: Keep Alive Target Functionality

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 12 : 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 33.

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

Response data:

Element	Datatype	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 <i>Transport Header</i> , 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 13 : 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		Query server result						
ulCookie	0xA55A5AA5	ulCookie	0xA55A5AA5					
ulLength	0	ulLength	42					
usChecksum	0	usChecksum	0					
usDatatype	0	usDatatype	0					
bDevice	0	bDevice	0					
bChannel	0	bChannel	0					
bSequenceNr	n	bSequenceNr	n					
bState	0	bState	0					
usTransactionID	m	usTransactionID	m					
ulReserved	0	ulReserved	0					
		ulStructVersion	1					
		szServerName	n	e	t	I	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 14 : 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 15 : 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 20) Note: If <i>ulError</i> is set, the following data will be suppressed.
ulDeviceCnt	DWORD	Number of supported devices

Table 16 : 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 20) Note: If <i>ulError</i> is set, the following data will be suppressed.
ulChannelCnt	DWORD	Number of supported channels

Table 17 : Administration - Query Number of Communication Channels - Result

Example:

Query channel count request		Query channel count result	
ulCookie	0xA55A5AA5	ulCookie	0xA55A5AA5
ulLength	6	ulLength	14
usChecksum	0	usChecksum	0
usDatatype	1	usDatatype	0
bDevice	0	bDevice	0
bChannel	0	bChannel	0
bSequenceNr	n	bSequenceNr	n
bState	0	bState	0
usTransactionID	m	usTransactionID	m
ulReserved	0	ulReserved	0
usDatatype	0x100	usDatatype	0x100
ulOption	1	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 20) Note: If <i>ulError</i> 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 18 : 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 20) Note: If <i>ulError</i> 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 19 : 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).

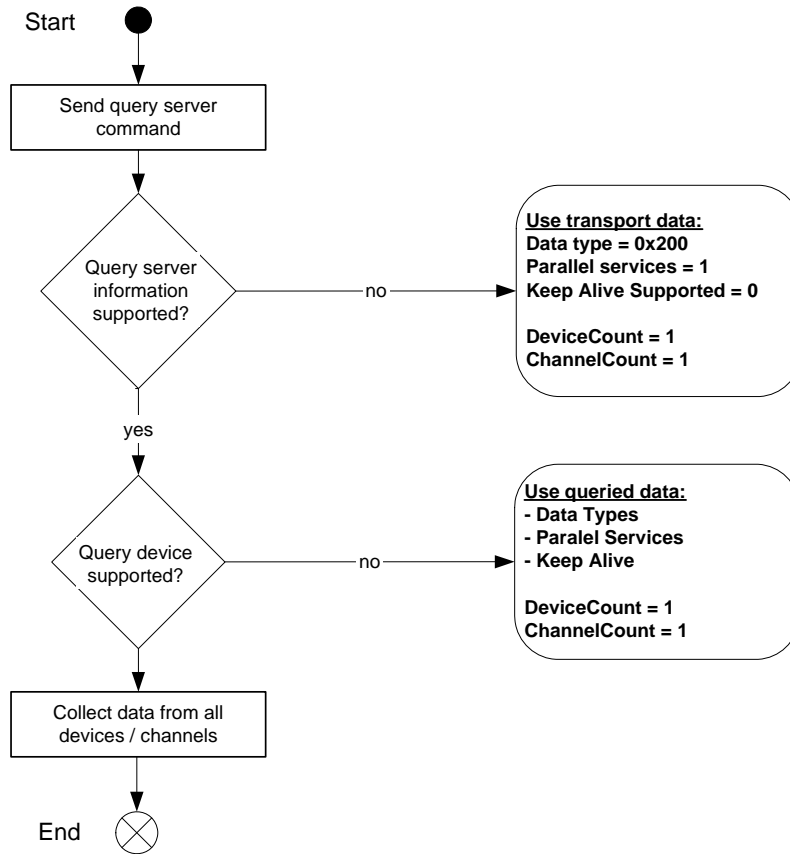


Figure 15: Fallback Behaviour

4.5 List of 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 <i>bDevice</i> is invalid
HIL_ADMIN_CHANNEL_NOTEXISTING	0x80000002	Channel is not existing on the device <i>bChannel</i> invalid

Table 20 : 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 21 : rcX Packet - Basic Structure

Note: Detailed description of rcX packet can be found in the '*netX DPM Interface Manual*' or in the communication protocol stack specific interface manual.

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 22 : 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).

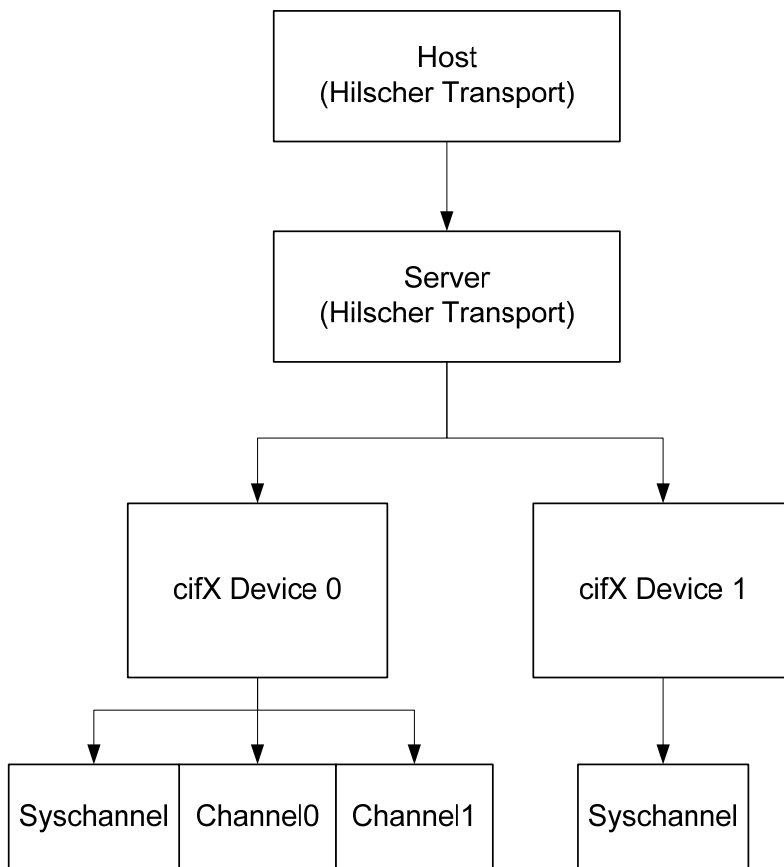


Figure 16 : rcX Packet Transmission Example

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 <i>bDevice</i>	Header field <i>bChannel</i>	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 23 : 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 '*Marshallled*' 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 <i>Marshaller</i> object
ulMethodId	DWORD	Method ID defining the function to execute (depends on the <i>Marshaller</i> object)
ulStatus	DWORD	Marshaller data frame state (see below)
ulError	DWORD	' <i>cifX Driver</i> ' 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 24 : 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_SYSTEMDEVICE	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 25 : *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
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

Method ID	Value	Description / cifX API Function Equivalent
MARSHALLER_CHANNEL_METHODID_HOSTSTATE	0x11	xChannelHostState
MARSHALLER_CHANNEL_METHODID_IOREAD	0x12	xChannelIORead
MARSHALLER_CHANNEL_METHODID_IOWRITE	0x13	xChannelIOWrite
MARSHALLER_CHANNEL_METHODID_IOREADSENDATA	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 26 : netXMarshaller Method ID Definitions

ulStatus bit 0 .. 15:

Bit	Description
0	0 = Result 1 = Request
1	1 = Sequence Number Support
2 ... 15	Reserved

ulStatus bit 16 ... 31:

Bit	Description
16 ... 31	Sequence Number (if supported, bit 1 set)

6.1 Classfactory Object - MethodID

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 27 : 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 28 : 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 29 : MARSHALLER_CF_METHODID_CREATEINSTANCE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	4
Data		
ulHandle	DWORD	Handle of the newly created object.

Table 30 : 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 31 : 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 32 : 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 33 : 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 34 : 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	<i>ulSize</i> parameter passed to <i>xDriverGetInformation()</i>

Table 35 : MARSHALLER_DRV_METHODID_GETINFO - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 36 : 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	<i>ulSize</i> parameter passed to <i>xDriverGetErrorDescription()</i>

Table 37 : MARSHALLER_DRV_METHODID_ERRORDESCR - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	ulHandle from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 38 : 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 39 : 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 40 : 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 41 : MARSHALLER_DRV_METHODID_ENUMCHANNELS - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 42 : 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 43 : MARSHALLER_DRV_METHODID_OPENCHANNEL - Request

Result:

Element	Datatype	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 44 : 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 netXMarshall 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 + <i>ulBoardNameSize</i>
Data		
ulBoardNameSize	DWORD	Length of the board name
szBoardname	CHAR[ulBoardNameSize]	Board name to open

Table 45 : MARSHALLER_DRV_METHODID_OPENSYSDEV - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 <code>xSystemdevice....</code>

Table 46 : 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 47 : MARSHALLER_SYSDEV_METHODID_CLOSE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 48 : 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 (see ' <i>cifX Device Driver</i> ' manual for details)
ulSize	DWORD	Size of the returned structure

Table 49 : MARSHALLER_SYSDEV_METHODID_INFO - Request

Result:

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

Table 50 : 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. netlC) 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 51 : MARSHALLER_SYSDEV_METHODID_RESET - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 52 : 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 53 : MARSHALLER_SYSDEV_METHODID_GETMBXSTATE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 54 : 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 + <i>ulSendSize</i>
Data		
ulSendSize	DWORD	Length of following message
abData	BYTE[ulSendSize]	Packet data to send
ulTimeout	DWORD	Timeout in ms for function call

Table 55 : MARSHALLER_SYSDEV_METHODID_PUTPACKET - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 56 : 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 57 : MARSHALLER_SYSDEV_METHODID_GETPACKET - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 58 : 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 59 : MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 60 : 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 61 : MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 62 : 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.

6.4 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 <code>MARSHALLER_DRV_METHODID_OPENCHANNEL</code>
ulMethodId	DWORD	1
ulStatus	DWORD	1 (+ Sequence Nr.)
ulError	DWORD	0
ulSize	DWORD	0

Table 63 : `MARSHALLER_CHANNEL_METHODID_CLOSE` - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 64 : `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 65 : MARSHALLER_CHANNEL_METHODID_GETMBXSTATE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 66 : 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 67 : MARSHALLER_CHANNEL_METHODID_PUTPACKET - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 68 : 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 69 : MARSHALLER_CHANNEL_METHODID_GETPACKET - Request

Result:

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

Table 70 : 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 71 : MARSHALLER_CHANNEL_METHODID_GETSENDPACKET - Request

Result:

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

Table 72 : 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 73 : MARSHALLER_CHANNEL_METHODID_CONFIGLOCK - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
ulState	DWORD	Returned State

Table 74 : 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 75 : MARSHALLER_CHANNEL_METHODID_RESET - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 76 : 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 77 : MARSHALLER_CHANNEL_METHODID_INFO - Request

Result:

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

Table 78 : 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 79 : MARSHALLER_CHANNEL_METHODID_WATCHDOG - Request

Result:

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

Table 80 : 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 81 : MARSHALLER_CHANNEL_METHODID_HOSTSTATE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	4
Data		
ulState	DWORD	Returned state

Table 82 : 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 83 : MARSHALLER_CHANNEL_METHODID_IOREAD - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	<i>ulDataLen</i> 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 84 : MARSHALLER_CHANNEL_METHODID_IOREAD - Result

6.4.13 MARSHALLER_CHANNEL_METHODID_IOWRITE

Corresponding cifX API function: *xChannellIOWrite()*

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 + <i>ulDataLen</i>
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 <i>abData</i>
abData	BYTE[ulDataLen]	Data to write

Table 85 : MARSHALLER_CHANNEL_METHODID_IOWRITE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 86 : 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 87 : MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	<i>ulDataLen</i> 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 88 : 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 ' <i>cifX Device Driver</i> ' manual)
ulState	DWORD	State to set (see ' <i>cifX Device Driver</i> ' manual)
ulTimeout	DWORD	Timeout in [ms]

Table 89 : MARSHALLER_CHANNEL_METHODID_BUSSTATE - Request

Result:

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

Table 90 : MARSHALLER_CHANNEL_METHODID_BUSSTATE - Result

6.4.16 MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK

Corresponding cifX API function: *xChannelControlBlock()*

Request: CIFS_CMD_READ_DATA Command (see 'cifX Device Driver' manual)

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 = CIFS_CMD_READ_DATA
ulOffset	DWORD	Offset in control block
ulDataLen	DWORD	Length of data to read

Table 91 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Request (READ_DATA)

Result:

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

Table 92 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Result (READ_DATA)

Request: CIFS_CMD_WRITE_DATA Command (see 'cifX Device Driver' manual)

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 + <i>ulDataLen</i>
Data		
ulCmd	DWORD	Command code = CIFS_CMD_WRITE_DATA
ulOffset	DWORD	Offset in control block
ulDataLen	DWORD	Length to write
abData	BYTE[ulDataLen]	Data to write

Table 93 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Request (WRITE_DATA)

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX Error code
ulSize	DWORD	0

Table 94 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Result (WRITE_DATA)

6.4.17 MARSHALLER_CHANNEL_METHODID_STATUSBLOCK

Corresponding cifX API function: *xChannelStatusBlock()*

Request: CIFS_CMD_READ_DATA Command (see 'cifX Device Driver' manual)

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 = CIFS_CMD_READ_DATA
ulOffset	DWORD	Offset in status block
ulDataLen	DWORD	Length to read

Table 95 : MARSHALLER_CHANNEL_METHODID_STATUSBLOCK - Request

Result:

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

Table 96 : MARSHALLER_CHANNEL_METHODID_STATUSBLOCK - Result

6.4.18 MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK

Corresponding cifX API function: *xChannelExtendedStatusBlock()*

Request: CFX_CMD_READ_DATA Command (see 'cifX Device Driver' manual)

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 = CFX_CMD_READ_DATA
ulOffset	DWORD	Offset in the extended status block
ulDataLen	DWORD	Length to read

Table 97 : MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK - Request

Result:

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

Table 98 : 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 99 : MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 100 : 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 101 : MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> 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 102 : 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: *xChannellIOInfo()*

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 103 : MARSHALLER_CHANNEL_METHODID_IOINFO - Request

Result:

Element	Data Type	Value
Marshaller Header		
ulHandle	DWORD	<i>ulHandle</i> from request
ulMethodId	DWORD	<i>ulMethodId</i> from request
ulStatus	DWORD	0 (+ Sequence Nr.)
ulError	DWORD	cifX error code
ulSize	DWORD	sizeof(CHANNEL_IO_INFORMATION)
Data		
tChannellInfo	CHANNEL_IO_INFORMATION	Channel IO information structure

Table 104 : MARSHALLER_CHANNEL_METHODID_IOINFO – Result

6.5 Createing 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 will be transparent 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_CREATEINSTANC E	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_CREATEINSTANC E	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	<i>Driver Handle</i>	ulHandle	<i>Driver Handle</i>
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	<i>Driver Handle</i>	ulHandle	<i>Driver Handle</i>
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	<i>Driver Handle</i>	ulHandle	<i>Driver Handle</i>
ulMethodID	8 MARSHALLER_DRV_METHODID_OPENCHANNEL	ulMethodID	8 MARSHALLER_DRV_METHODID_OPENCHANNEL
ulSize	13	ulSize	4
ulBoardName	5	ulData	<i>Channel Handle</i>
DeviceName	"cifX0"		

- Call *xChannelGetMBXState* on the system device object

Request Data		Returned Data	
ulHandle	<i>Channel Handle</i>	ulHandle	<i>Channel Handle</i>
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

1		Returned Data	
ulHandle	<i>Channel Handle</i>	ulHandle	<i>Channel Handle</i>
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 xChannellOWrite

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 xChannellOWrite

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 List of Tables

Table 1: List of Revisions	4
Table 2: Terms, Abbreviations and Definitions	5
Table 3: References	5
Table 4: Transport Header Definition	14
Table 5 : Mandatory Data Types	15
Table 6: Feature Data Type	15
Table 7 : State Definitions	15
Table 8: Keep Alive - Faulty Request	18
Table 9: Keep Alive Request	19
Table 10: Keep Alive Acknowledge	20
Table 11: Keep Alive Response	20
Table 12 : Administration Data Types	23
Table 13 : Administration - Query Server Information Result	24
Table 14 : Administration - Query Device Information - Basic Request	26
Table 15 : Administration - Query Device Information - Options List	26
Table 16 : Administration - Query Number of Devices - Result	27
Table 17 : Administration - Query Number of Communication Channels - Result	28
Table 18 : Administration - Query Device Information - Result	29
Table 19 : Administration - Query Channel Information - Result	31
Table 20 : Error Codes for Administration Commands	34
Table 21 : rcX Packet - Basic Structure	35
Table 22 : rcX Packet Transfer (Channel/Device mapping)	35
Table 23 : rcX Packet - Addressing Example	36
Table 24 : netXMarshaller Data Frame Header	37
Table 25 : netXMarshaller Handle Definitions	38
Table 26 : netXMarshaller Method ID Definitions	40
Table 27 : MARSHALLER_CF_METHODID_SERVERVERSION - Request	41
Table 28 : MARSHALLER_CF_METHODID_SERVERVERSION - Result	41
Table 29 : MARSHALLER_CF_METHODID_CREATEINSTANCE - Request	42
Table 30 : MARSHALLER_CF_METHODID_CREATEINSTANCE - Result	42
Table 31 : MARSHALLER_DRV_METHODID_OPEN - Request	43
Table 32 : MARSHALLER_DRV_METHODID_OPEN - Result	43
Table 33 : MARSHALLER_DRV_METHODID_CLOSE - Request	44
Table 34 : MARSHALLER_DRV_METHODID_CLOSE - Result	44
Table 35 : MARSHALLER_DRV_METHODID_GETINFO - Request	45
Table 36 : MARSHALLER_DRV_METHODID_GETINFO - Result	45
Table 37 : MARSHALLER_DRV_METHODID_ERRORDESCR - Request	46
Table 38 : MARSHALLER_DRV_METHODID_ERRORDESCR - Result	46
Table 39 : MARSHALLER_DRV_METHODID_ENUMBOARDS - Request	47
Table 40 : MARSHALLER_DRV_METHODID_ENUMBOARDS - Result	47
Table 41 : MARSHALLER_DRV_METHODID_ENUMCHANNELS - Request	48
Table 42 : MARSHALLER_DRV_METHODID_ENUMCHANNELS - Result	48
Table 43 : MARSHALLER_DRV_METHODID_OPENCHANNEL - Request	49
Table 44 : MARSHALLER_DRV_METHODID_OPENCHANNEL - Result	49
Table 45 : MARSHALLER_DRV_METHODID_OPENSYSDEV - Request	50
Table 46 : MARSHALLER_DRV_METHODID_OPENSYSDEV - Result	50
Table 47 : MARSHALLER_SYSDEV_METHODID_CLOSE - Request	51
Table 48 : MARSHALLER_SYSDEV_METHODID_CLOSE - Result	51
Table 49 : MARSHALLER_SYSDEV_METHODID_INFO - Request	52
Table 50 : MARSHALLER_SYSDEV_METHODID_INFO - Result	52
Table 51 : MARSHALLER_SYSDEV_METHODID_RESET - Request	53
Table 52 : MARSHALLER_SYSDEV_METHODID_RESET - Result	53
Table 53 : MARSHALLER_SYSDEV_METHODID_GETMBXSTATE - Request	54
Table 54 : MARSHALLER_SYSDEV_METHODID_GETMBXSTATE - Result	54
Table 55 : MARSHALLER_SYSDEV_METHODID_PUTPACKET - Request	55
Table 56 : MARSHALLER_SYSDEV_METHODID_PUTPACKET - Result	55
Table 57 : MARSHALLER_SYSDEV_METHODID_GETPACKET - Request	56
Table 58 : MARSHALLER_SYSDEV_METHODID_GETPACKET - Result	56
Table 59 : MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE - Request	58
Table 60 : MARSHALLER_SYSDEV_METHODID_FINDFIRSTFILE - Result	58
Table 61 : MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE - Request	59
Table 62 : MARSHALLER_SYSDEV_METHODID_FINDNEXTFILE - Result	59
Table 63 : MARSHALLER_CHANNEL_METHODID_CLOSE - Request	61

Table 64 : MARSHALLER_CHANNEL_METHODID_CLOSE - Result	61
Table 65 : MARSHALLER_CHANNEL_METHODID_GETMBXSTATE - Request	62
Table 66 : MARSHALLER_CHANNEL_METHODID_GETMBXSTATE - Result	62
Table 67 : MARSHALLER_CHANNEL_METHODID_PUTPACKET - Request	63
Table 68 : MARSHALLER_CHANNEL_METHODID_PUTPACKET - Result	63
Table 69 : MARSHALLER_CHANNEL_METHODID_GETPACKET - Request	64
Table 70 : MARSHALLER_CHANNEL_METHODID_GETPACKET - Result	64
Table 71 : MARSHALLER_CHANNEL_METHODID_GETSENDPACKET - Request	65
Table 72 : MARSHALLER_CHANNEL_METHODID_GETSENDPACKET - Result	65
Table 73 : MARSHALLER_CHANNEL_METHODID_CONFIGLOCK - Request	66
Table 74 : MARSHALLER_CHANNEL_METHODID_CONFIGLOCK - Result	66
Table 75 : MARSHALLER_CHANNEL_METHODID_RESET - Request	67
Table 76 : MARSHALLER_CHANNEL_METHODID_RESET - Result	67
Table 77 : MARSHALLER_CHANNEL_METHODID_INFO - Request	68
Table 78 : MARSHALLER_CHANNEL_METHODID_INFO - Result	68
Table 79 : MARSHALLER_CHANNEL_METHODID_WATCHDOG - Request	69
Table 80 : MARSHALLER_CHANNEL_METHODID_WATCHDOG - Result	69
Table 81 : MARSHALLER_CHANNEL_METHODID_HOSTSTATE - Request	70
Table 82 : MARSHALLER_CHANNEL_METHODID_HOSTSTATE - Result	70
Table 83 : MARSHALLER_CHANNEL_METHODID_IOREAD - Request	71
Table 84 : MARSHALLER_CHANNEL_METHODID_IOREAD - Result	71
Table 85 : MARSHALLER_CHANNEL_METHODID_IOWRITE - Request	72
Table 86 : MARSHALLER_CHANNEL_METHODID_IOWRITE - Result	72
Table 87 : MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA - Request	73
Table 88 : MARSHALLER_CHANNEL_METHODID_IOREADSENDDATA - Result	73
Table 89 : MARSHALLER_CHANNEL_METHODID_BUSSTATE - Request	74
Table 90 : MARSHALLER_CHANNEL_METHODID_BUSSTATE - Result	74
Table 91 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Request (READ_DATA)	75
Table 92 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Result (READ_DATA)	75
Table 93 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Request (WRITE_DATA)	76
Table 94 : MARSHALLER_CHANNEL_METHODID_CONTROLBLOCK - Result (WRITE_DATA)	76
Table 95 : MARSHALLER_CHANNEL_METHODID_STATUSBLOCK - Request	77
Table 96 : MARSHALLER_CHANNEL_METHODID_STATUSBLOCK - Result	77
Table 97 : MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK - Request	78
Table 98 : MARSHALLER_CHANNEL_METHODID_EXTSTATUSBLOCK - Result	78
Table 99 : MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE - Request	79
Table 100 : MARSHALLER_CHANNEL_METHODID_FINDFIRSTFILE - Result	79
Table 101 : MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE - Request	80
Table 102 : MARSHALLER_CHANNEL_METHODID_FINDNEXTFILE - Result	80
Table 103 : MARSHALLER_CHANNEL_METHODID_IOINFO - Request	82
Table 104 : MARSHALLER_CHANNEL_METHODID_IOINFO - Result	82

7.2 List of Figures

Figure 1: Overview	8
Figure 2 : Software Structure	9
Figure 3: Internal Block Diagram	10
Figure 4 : Internal Data Flow	11
Figure 5 : Data Encapsulation	14
Figure 6: Keep Alive - First Request	16
Figure 7: Keep Alive - Continuing	17
Figure 8: Keep Alive Host Functionality	21
Figure 9: Keep Alive Target Functionality	22
Figure 10 : Query Server Information - Example	25
Figure 11 : Administration - Query Number of Devices - Example	27
Figure 12 : Administration - Query Number of Communication Channel - Example	28
Figure 13 : Query Device Information - Example	30
Figure 14 : Query Channel Information - Example	32
Figure 15: Fallback Behaviour	33
Figure 16 : rcX Packet Transmission Example	36

7.3 Contacts

Headquarters

Germany

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

Support

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

Subsidiaries

China

Hilscher Ges.f.Systemaut. mbH
Shanghai Representative Office
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support

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

France

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

Support

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

India

Hilscher India Pvt. Ltd.
New Delhi - 110 025
Phone: +91 11 40515640
E-Mail: info@hilscher.in

Italy

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

Support

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

Japan

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

Support

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

Korea

Hilscher Korea Inc.
Suwon-Si, 443-810
Phone: +82-31-204-6190
E-Mail: info@hilscher.kr

Switzerland

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

Support

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

USA

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

Support

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