Protocol API

Ethernet

Packet interface

V4.4.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC060901API09EN | Revision 9 | English | 2017-05 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This manual describes the application interface of the Ethernet protocol stack implementation. Use this manual to support and guide you through the integration process of the given stack into your own application.

This stack was developed based upon Hilscher's Task Layer Reference Programming Model. This programming model is a description of how to develop a task in general, which is a convention defining a combination of appropriate functions belonging to the same task. Furthermore, it defines how different tasks have to communicate with each other in order to exchange their data. The Reference Model is commonly used by all developers at Hilscher and shall be used by you as well when writing your application task on top of the stack.

1.1.1 List of Revisions

Rev	Date	Name	Revisions
6	2011-04-14	YZ	Minor revision. Formatting and clean up. All error and status codes are now in the chapter
7	2011-09-27	AM	Reworked manual for Version V4.1.0.0. Removed all pre V4.1.0.0 related services.
8	2013-11-28	AM	Update of manual for V4.2.x.x Versions of Ethernet Interface. Including Description different operational modes and related APIs.
9	2017-05-31	AM/RG/ MB/BM	Version 4.4.0 Document restructured. Description of `ETH_INTF_GET_HW_STAT_REQ.` Behavior of BusOn/Off described. Added services `ETH_INTF_SET_MULTICAST_SINGLE_REQ` and `ETH_INTF_CLR_MULTICAST_SINGLE_REQ`. Added more text to section TCP/UDP Protocol API

Table 1: List of Revisions

1.2 System Requirements

This software package has the following system requirements to its environment:

- netX-Chip as CPU hardware platform
- operating system for task scheduling required
- operating system rcX currently required

1.3 Intended Audience

This manual is suitable for software developers with the following background:

- Knowledge of the C programming language
- Knowledge of the use of the real-time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model
- Knowledge of Ethernet (IEEE 802.3)

1.4 Specifications

The data below applies to the Ethernet firmware and stack version V4.4.0.

The firmware/stack is based on the Ethernet specification (IEEE 802.3).

Technical Data

Maximum frame length	1518 Bytes, including source and target addresses and Ethertype
Size of receive/transmit queue	4 telegrams each
Modes	OEM mode: the same MAC Address as the Industrial Ethernet protocol is used Ethernet (NDIS) mode: a different MAC Address than that of the Industrial Ethernet protocol is used Availability of these modes depends on the Industrial Ethernet stack
Baud rates	10 and 100 MBit/s
Data transport layer	Ethernet II, IEEE 802.3

Features

- Transmission and reception of broadcast messages
- TCP/IP support (depending on the Industrial Ethernet stack used in conjunction with the Ethernet stack V4.4.0).

Configuration

By configuration packets to transfer warm start parameters (in OEM mode).

Diagnostic

Firmware supports common and extended diagnostic in the dual-port-memory, see section 5.1“ *Extended Status*” on page 47.

1.5 Terms, Abbreviations and Definitions

Term	Description
AP (-task)	Application (-task) on top of the stack
ETH_INTF (task)	Ethernet protocol task
DPM	Dual Port Memory
EDD	Ethernet Device Driver
FIFO	First in first out
IANA	Internet Assigned Numbers Authority
NDIS	Network Driver Interface Specification
Rx	Receive
TCPIP (task)	TCP/IP protocol interface (task)
Tx	Transmit
Industrial Ethernet protocol	The (main) Real-Time Ethernet protocol running on the netX

Table 2: Terms, Abbreviations and Definitions

1.6 References to Documents

This document based on the following specification respectively documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX based products, Revision 12, English, 2012.

[2] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, TCP/IP Packet interface, Revision 14, English, 2017.

Table 3: References

1.7 Legal Notes

Copyright

Important notes

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

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

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

Liability disclaimer

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

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

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

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

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

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

Warranty

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

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

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

Costs of support, maintenance, customization and product care

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

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby

the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

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

Confidentiality

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

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

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

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

Export provisions

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

2 Getting Started

This section explains some essential information you should know when starting to work with the Ethernet Protocol API.

2.1 Features / Changes

Stack version	Features / Changes
1.x.x.x	Raw Ethernet access
2.x.x.x	Raw Ethernet access + TCPIP task packet router. The layout of the Set Configuration Request packet has changed. Additionally the startup behavior was adapted to definitions in DPM manual. (Set Configuration + Channel Init is now needed) For backwards compatibility the stack supports the old 1.x Set Configuration packet and emulates the old startup behavior.
4.0.x.x	Support for One-Port Interfaces, adapted to new internal rcX API
4.1.x.x	Complete rework of the API . (Incompatible to previous versions) Provide a full Raw Ethernet Interface by using a dedicated MAC Address. Changed Task Protocol Class. Removed unneeded services. Simplified Packet API .
4.2.x.x	The API was again updated. The Ethernet protocol task can now run in two different operation modes. In OEM mode the API is almost compatible with version 4.0.x.x. In Ethernet (NDIS) mode a new API was introduced.
4.4.x.x	In Ethernet (NDIS) mode a new API to enable / disable receipt of multicast frames was added

Table 4: Overview about features of and changes between different Ethernet Protocol versions

2.2 Operation Modes

The Ethernet protocol task offers two different operation modes: *OEM* and *Ethernet (NDIS)*.

Note: Availability of these modes depends on the Industrial Ethernet stack!

The API to be applied depends on the chosen operation mode. The host application can detect the current operation mode by evaluating the Protocol Class Field of the Channel Information. The protocol class will be set depending on the operation mode to the following values.

Constant	Value	Description
RCX_PROT_CLASS_OEM	0xFFF0	The Ethernet protocol task is running in operation mode OEM.
RCX_PROT_CLASS_ETHERNET	0x0024	The Ethernet protocol task is running in operation mode Ethernet (NDIS)

Table 5: Ethernet protocol task operation modes

Information on how to access the protocol class field of the channel information can be found in the *Dual-Port Memory Interface Manual for netX based Products* [1].

The operation mode cannot be configured by the host application. It is specified by the firmware. In some cases the operation mode might be changeable by means of firmware tag list.

The Ethernet Protocol API depends on the operation mode and can be split into three parts:

- **Ethernet Protocol API in OEM mode**: This API is only available if OEM operation mode is active. In OEM mode the Ethernet interface uses the same MAC Address as the Industrial Ethernet protocol running within the netX. Thus the host application is available to the network with the same identity as the Industrial Ethernet protocol. This includes TCP/IP parameters, if applicable. Of course, if the firmware runs an internal TCP/IP Stack, internal filtering might be applied to Ethernet frames sent or received by the host application depending on the internal TCP/IP state. If applicable, the host application's TCP/IP implementation must use the same TCP/IP parameters as the firmware's internal TCP/IP stack.
- **Ethernet Protocol API in Ethernet (NDIS) mode**: This API is only available if Ethernet (NDIS) operation mode is active. In Ethernet (NDIS) mode, the Ethernet Interface uses a different MAC Address than the Industrial Ethernet protocol running within the netX. For this purpose, the fourth MAC Address defined in the Security Memory is used. The host application will be visible to the Network with this dedicated MAC Address and thus appear as an individual device. Therefore any Ethernet Protocol parameter can be chosen freely with respect to the Protocol Standard and Definitions. The host application must ensure, that no parameter conflicts can occur. In particular, it must never use the same IP address as the Industrial Ethernet protocol. In order to protect the Industrial Ethernet protocol operation, the Ethernet Switch might filter certain Ethernet Frames send or received by the host application. Details of this filtering depend on the particular Firmware and are described in the corresponding manual.
- **TCP/IP Protocol API**: This API might be available in both Ethernet protocol operation modes. The availability depends on the particular firmware and is described in the corresponding firmware/protocol manual. The API provides access to the firmware's internal TCP/IP stack reusing its API. Thus, this API is described in the TCP/IP Protocol Manual. The Ethernet protocol task might apply special filtering to packets of this API to prevent reconfigurations of the internal TCP/IP stack. This setting depends on the particular firmware. Details can be found in the corresponding firmware/protocol manual. As the internal TCP/IP stack is used, the host application is visible to the network with the same network parameters

as the Industrial Ethernet protocol. (In particular IP address settings) For proper Industrial Ethernet protocol operation it is essential, that the host application handles all TCP/IP indications properly. Most important to note are the TCP/IP shutdown indication on which the host application is required to immediately close all open sockets.

Note: The packet structure definitions from the public header file and the command code definitions of the different modes use very similar spellings for send and receive commands. Be carefully to choose the correct definitions when programming the API.

2.2.1 Compatibility of Ethernet Protocol Stack V4.2 to older Versions

As explained above, the Ethernet protocol stack implements two different operation modes since version V4.2.0.0. The availability of these modes depends on the firmware and may be configured by means of the firmware tag list. These modes are:

- OEM Mode: This is the original mode as supported by the previous versions. Depending on the firmware, either the Ethernet Frames are directly passed between the Ethernet protocol task and the netX Ethernet switch or they are filtered within the firmware's internal TCP/IP Stack. The default netX Ethernet MAC address will be used. In this operation mode, the task is compatible to older versions concerning the API.
- Ethernet (NDIS) Mode: This mode provides network access using a dedicated Ethernet MAC address. Frames are directly passed to the switch and routed either to the netX firmware itself or to the network depending on the destination MAC address. The underlying switch might apply filtering to the frames to protect the Industrial Ethernet protocol from any disturbance due to frames sent or received by the host application. This mode uses a new and simplified API.

Additionally, the Ethernet protocol task might provide access to the internal TCP/IP Stack integrated within the netX firmware. The availability of any of these features depends on the respective firmware and is documented within the corresponding protocol manuals.

2.2.2 Protocol Parameters for OEM Mode

The following protocol parameters can be configured in OEM mode (using the *Set Configuration Service*) if allowed by protocol stack:

Parameter Auto-Negotiation Port 0

The `Auto-Negotiation Port 0` parameter holds the auto-negotiation mode for Ethernet port 0. Auto-Negotiation means: An interface with Auto-Negotiation automatically determines a set of correct communication parameters if the interface it communicates with is also capable of Auto-Negotiation.

Parameter Auto-Negotiation Port 1

The `Auto-Negotiation Port 1` parameter holds the auto-negotiation mode for Ethernet port 1.

Parameter Duplex-Mode Port 0

The `Duplex Mode Port 0` parameter holds the duplex-mode for Ethernet port 0.

Parameter Duplex-Mode Port 1

The `Duplex Mode Port 1` parameter holds the duplex-mode for Ethernet port 1.

Parameter Transmission Rate Port 0

The `Transmission Rate Port 0` parameter holds the transmission-rate (10/100 MBit/s) for Ethernet port 0.

Parameter Transmission Rate Port 1

The `Transmission Rate Port 1` parameter holds the transmission-rate (10/100 MBit/s) for Ethernet port 0.

Parameter Auto Cross-Over Port0

The `Auto Cross-Over Port 0` parameter holds the cross-over configuration (auto cross-over on/off) for Ethernet port 0.

Auto cross-over means: An interface with Auto-Crossover capability will automatically detect and correct if the data lines have been exchanged vice versa.

Parameter Auto Cross-Over Port1

The `Auto Cross-Over Port 1` parameter holds the cross-over configuration (auto cross-over on/off) for Ethernet port 1.

Parameter Reserved 1 Port 0

Reserved parameter for Ethernet port 0.

Parameter Reserved 1 Port 1

Reserved parameter for Ethernet port 1.

Parameter Reserved 2 Port 0

Reserved parameter for Ethernet port 0.

Parameter Reserved 2 Port 1

Reserved parameter for Ethernet port 1.

2.3 Start-up of the Ethernet Interface Task

After power on or reset the device performs a start-up initialization. During this phase several steps are taken to bring the device from uninitialized state to operation. First, the hardware will be self-tested and the internal operating system will be started. Then the Ethernet protocol task will be started. The Ethernet protocol task will setup its assigned DPM Channel, setup its API according the operation mode and establish a connection to TCP/IP stack if necessary. Afterwards, the Ethernet Protocol is ready for use by the user's application.

3 The Application Interface

This chapter defines the application interface of the stack. The application itself uses the dual-port interface as described in *Dual-Port Memory Interface Manual for netX based Products* [1].

3.1 Ethernet Protocol API in OEM Mode

The following packets are available when using the Ethernet Protocol API in OEM mode:

Overview over the Packets of the Ethernet Stack (OEM Mode)			
Section	**Packet**	**Command code**	**Page**
3.1.2	Set Configuration Request	0x3B00	15
	Set Configuration Confirmation	0x3B01	17
3.1.3	Send Ethernet Frame Request	0x3B02	19
	Send Ethernet Frame Confirmation	0x3B03	21
3.1.4	Received Ethernet Frame Indication	0x3B04	22
	Received an Ethernet Frame Response	0x3B05	24
3.1.5	Ethernet Register Application Request	0x3B06	25
	Ethernet Register Application Confirmation	0x3B07	27
3.1.6	IP Configuration Indication	0x3B08	28
	IP Configuration Response	0x3B09	30
3.1.7	Get Hardware Statistics Request	0x3BFC	31
	Get Hardware Statistics Confirmation	0x3BFD	32

Table 6: Overview over the Configuration Packets of the Ethernet Protocol Stack

The following topics have to be taken into account when using the Ethernet Protocol API in OEM mode:

1. In this operation mode the Ethernet Protocol must be configured using the Set Configuration Service prior to sending and receiving Ethernet frames.
2. The new configuration must be applied using the generic RCX_CHANNEL_INIT_REQ service afterwards.
3. In order to send or receive Ethernet frames, the host application must register using the generic RCX_REGISTER_APP_REQ service.
4. The same MAC Address as in the Industrial Ethernet protocol running within the netX has to be applied, both protocols share the same identity. If applicable, the host application's TCP/IP implementation must use the same TCP/IP parameters as the firmware's internal TCP/IP stack.

3.1.1 Common Services

The Ethernet protocol task supports the following common services which are described in the *Dual-Port Memory Interface Manual for netX based Products* [1]:

- **Register Application Service**: used to register application.
- **Unregister Application Service**: used to unregister application.
- **Channel Initialization Service**: used to initialize configuration set by Set Configuration Service
- **Identify Firmware Service**: used to retrieve identification information from Ethernet protocol task.
- **Delete Configuration Service**: used to delete configuration of Ethernet protocol task.
- **Start/Stop Communication Service**: used to enable or disable communication. This is mainly needed for Ethernet Interface Task Running in OEM operation mode.
- **Lock/Unlock configuration Service**: used to lock or unlock Ethernet protocol task configuration.

Important: If the Ethernet Interface is running in OEM mode and the application is using raw Ethernet mode it is essential to activate the bus either using `RCX_START_STOP_COMM_REQ` or via dual port memory flags.

3.1.2 Set Configuration Service

The Set Configuration Service shall be used to configure the Ethernet protocol task. Depending on the Firmware, a Set Configuration Service with parameters deviating from default values might be rejected to prevent impact on Industrial Ethernet protocol operation. Furthermore a Set Configuration Request with invalid parameters or field `ulModeFlags` set to disabled APIs will also be rejected. After a successful Set Configuration Confirmation from netX, the host application shall issue a Channel Init to load the new configuration.

3.1.2.1 Set Configuration Request

Packet Structure Reference

```
/*****************************************************************************/
/*             Set config request                                            */
/*****************************************************************************/

typedef struct ETH_INTF_SET_CONFIG_REQ_DATA_Ttag  ETH_INTF_SET_CONFIG_REQ_DATA_T;

#define ETH_INTF_SET_CONFIG_REQ_FLAG_USE_ETHERNET    0x00000001
#define ETH_INTF_SET_CONFIG_REQ_FLAG_USE_TCPIP       0x00000002

struct ETH_INTF_SET_CONFIG_REQ_DATA_Ttag
{
  TLR_UINT32  ulAutoNegotiationPort0;
  TLR_UINT32  ulAutoNegotiationPort1;

  TLR_UINT32  ulDuplexMode0;
  TLR_UINT32  ulDuplexMode1;

  TLR_UINT32  ulTransmissionRatePort0;
  TLR_UINT32  ulTransmissionRatePort1;

  TLR_UINT32  ulAutoCrossOverPort0;
  TLR_UINT32  ulAutoCrossOverPort1;

  TLR_UINT32  ulReserved1Port0;
  TLR_UINT32  ulReserved1Port1;

  TLR_UINT32  ulReserved2Port0;
  TLR_UINT32  ulReserved2Port1;

  TLR_UINT32  ulModeFlags;
};

typedef struct ETH_INTF_SET_CONFIG_REQ_PCK_Ttag ETH_INTF_SET_CONFIG_REQ_PCK_T;

struct  ETH_INTF_SET_CONFIG_REQ_PCK_Ttag
{
  /** Packet header */
  TLR_PACKET_HEADER_T             tHead;
  /** Packet data */
  ETH_INTF_SET_CONFIG_REQ_DATA_T  tData;
};
```

Packet Description

Structure ETH_INTF_SET_CONFIG_REQ_PCK_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the ETH_INTF-Task process queue
ulSrc	UINT32	0	Source Queue-Handle. Set to zero
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32	arbitrary	Source End Point Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulLen	UINT32	52	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Unused in requests. Set to zero.
ulCmd	UINT32	0x3B00	ETH_INTF_SET_CONFIG_REQ - Command
ulExt	UINT32	0	Extension. Set to zero.
ulRout	UINT32	0	Routing. Set to zero.
tData - Structure ETH_INTF_SET_CONFIG_REQ_DATA_T			
ulAutoNegotiationPort0	UINT32	0 1	Deactivate Auto-Negotiation Port 0 Activate Auto-Negotiation for Port 0 (default)
ulAutoNegotiationPort1	UINT32	0 1	Deactivate Auto-Negotiation Port 1 Activate Auto-Negotiation for Port 1 (default)
ulDuplexMode0	UINT32	0 1	Set Port 0 to Half-Duplex Mode Set Port 0 to Full-Duplex Mode (default)
ulDuplexMode1	UINT32	0 1	Set Port 1 to Half-Duplex Mode Set Port 1 to Full-Duplex Mode (default)
ulTransmissionRatePort0	UINT32	10 100	Set Port 0 to 10 MBit/s Transmission Rate Set Port 0 to 100 MBit/s Transmission Rate (default)
ulTransmissionRatePort1	UINT32	10 100	Set Port 1 to 10 MBit/s Transmission Rate Set Port 1 to 100 MBit/s Transmission Rate (default)
ulAutoCrossOverPort0	UINT32	0 1	Disable Port 0 Auto cross-over Enable Port 0 Auto cross-over (default)
ulAutoCrossOverPort1	UINT32	0 1	Disable Port 0 Auto cross-over Enable Port 0 Auto cross-over (default)
ulReserved1Port0	UINT32	0	Reserved for further use, Set to zero
ulReserved1Port1	UINT32	0	Reserved for further use, Set to zero
ulReserved2Port0	UINT32	0	Reserved for further use, Set to zero
ulReserved2Port1	UINT32	0	Reserved for further use, Set to zero
ulModeFlags	UINT32	0-3	Bit Mask to select used Interfaces: 0x00000001: Enable raw Ethernet Interface 0x00000002: Enable TCP/UDP Interface

Table 7: ETH_INTF_SET_CONFIG_REQ_T –Request Packet to set the configuration

3.1.2.2 Set Configuration Confirmation

Packet Structure Reference

```
/*****************************************************************************/
/*             Set configuration comfirmation                                */
/*****************************************************************************/

/* set configuration confirmation data */
typedef struct ETH_INTF_SET_CONFIG_CNF_DATA_Ttag ETH_INTF_SET_CONFIG_CNF_DATA_T;

#define ETH_INTF_SET_CONFIG_CNF_ETHERNET_STATUS_UNDEF       0x00000000
#define ETH_INTF_SET_CONFIG_CNF_ETHERNET_STATUS_UNAVAIL     0x00000001
#define ETH_INTF_SET_CONFIG_CNF_ETHERNET_STATUS_DISABLED    0x00000002
#define ETH_INTF_SET_CONFIG_CNF_ETHERNET_STATUS_RESTRICTED  0x00000003
#define ETH_INTF_SET_CONFIG_CNF_ETHERNET_STATUS_ENABLED     0x00000004

#define ETH_INTF_SET_CONFIG_CNF_TCPIP_STATUS_UNDEF       0x00000000
#define ETH_INTF_SET_CONFIG_CNF_TCPIP_STATUS_UNAVAIL     0x00000001
#define ETH_INTF_SET_CONFIG_CNF_TCPIP_STATUS_DISABLED    0x00000002
#define ETH_INTF_SET_CONFIG_CNF_TCPIP_STATUS_RESTRICTED  0x00000003
#define ETH_INTF_SET_CONFIG_CNF_TCPIP_STATUS_ENABLED     0x00000004

struct ETH_INTF_SET_CONFIG_CNF_DATA_Ttag
{
  TLR_UINT32  ulStatusEthernet;
  TLR_UINT32  ulStatusTcpIp;
};

typedef struct ETH_INTF_SET_CONFIG_CNF_PCK_Ttag ETH_INTF_SET_CONFIG_CNF_PCK_T;

struct  ETH_INTF_SET_CONFIG_CNF_PCK_Ttag
{
  /** packet header */
  TLR_PACKET_HEADER_T             tHead;
  /** packet data */
  ETH_INTF_SET_CONFIG_CNF_DATA_T  tData;
};
```

Packet Description

Structure ETH_INTF_SET_CONFIG_CNF_PCK_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32		Source End Point Identifier. Will contain same value as in Request.
ulLen	UINT32	8	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Will contain same value as in the request.
ulSta	UINT32	0	Packet Status. Will be set to nonzero error code if service failed.
ulCmd	UINT32	0x00003B01	ETH_INTF_SET_CONFIG_CNF - Command
ulExt	UINT32	0	Extension. Ignore.
ulRout	UINT32	0	Routing. Ignore
tData - Structure ETH_INTF_SET_CONFIG_CNF_DATA_T			
ulStatusEthernet	UINT32	0 1 2 3 4 5	Reserved Ethernet Interface is not available. Ethernet Interface is not enabled by host application Ethernet Interface is enabled but running in restricted mode Ethernet Interface is enabled Ethernet Interface is enabled and using a dedicated MAC Address (reserved for future use)
ulStatusTcpIp	UINT32	0 1 2 3 4	Reserved TCP/IP API not available TCP/IP API is not enabled by host application TCP/IP API is enabled but running in restricted mode TCP/IP API is enabled

Table 8: ETH_INTF_SET_CONFIG_CNF_T –Confirmation Packet of Set Configuration Service

3.1.3 OEM Mode Send Ethernet Frame Service

This service shall be used to send a frame to Ethernet. This service can only be used if the Ethernet protocol task had been successfully configured, the configuration had been initialized and the host application had registered. The minimum length of an Ethernet frame is 60 bytes and the maximum is 1518 Bytes.

3.1.3.1 Send Ethernet Frame Request

Packet Structure Reference

```
/*****************************************************************************/
/*             Send Ethernet frame request                                   */
/*****************************************************************************/
#define ETH_INTF_MIN_SEND_ETH_FRAME_SIZE    60  /* Minimum data count for request      */

#define ETH_INTF_MAX_SEND_ETH_FRAME_SIZE  1518  /* Maximum data count for request      */

#define ETH_INTF_SEND_ETH_FRAME_HEADER_SIZE   8 /* 8 Bytes for header (port and reserved)
*/
#define ETH_INTF_SEND_ETH_FRAME_RSRV_SIZE     4 /* 4 Bytes reserved in send request    */

#define ETH_INTF_SEND_ETH_FRAME_DEFAULT_PORT  0x00000000L /* Default port for 2 switch */
#define ETH_INTF_SEND_ETH_FRAME_PORT_0        0x00000001L /* Port 0                    */
#define ETH_INTF_SEND_ETH_FRAME_PORT_1        0x00000002L /* Port 1                    */

#define ETH_INTF_MIN_SEND_ETH_FRAME_PACKET_LENGTH \
  ETH_INTF_MIN_SEND_ETH_FRAME_SIZE + ETH_INTF_SEND_ETH_FRAME_HEADER_SIZE

#define ETH_INTF_MAX_SEND_ETH_FRAME_PACKET_LENGTH \
  ETH_INTF_MAX_SEND_ETH_FRAME_SIZE + ETH_INTF_SEND_ETH_FRAME_HEADER_SIZE

/** Send Ethernet frame request data*/
typedef struct ETH_INTF_SEND_ETH_FRAME_REQ_DATA_Ttag  ETH_INTF_SEND_ETH_FRAME_REQ_DATA_T;

/** Send Ethernet frame request data
 *
 * The structure represents the specific request parameter of the service.
 */
struct ETH_INTF_SEND_ETH_FRAME_REQ_DATA_Ttag
{
  TLR_UINT32 ulEthPort;
  TLR_UINT32 ulReserved;

  TLR_UINT8  abData[ETH_INTF_MAX_SEND_ETH_FRAME_SIZE];
};

/** Send Ethernet frame request */
typedef struct ETH_INTF_SEND_ETH_FRAME_REQ_PCK_Ttag ETH_INTF_SEND_ETH_FRAME_REQ_PCK_T;

struct  ETH_INTF_SEND_ETH_FRAME_REQ_PCK_Ttag
{
  /** Packet header */
  TLR_PACKET_HEADER_T                tHead;
  /** Packet data */
  ETH_INTF_SEND_ETH_FRAME_REQ_DATA_T  tData;
};
```

Packet Description

Structure ETH_INTF_SEND_ETH_FRAME_REQ_PCK_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the ETH_INTF-Task process queue
ulSrc	UINT32	0	Source Queue-Handle. Set to zero
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32	arbitrary	Source End Point Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulLen	UINT32	68 - 1526	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Unused in requests. Set to zero.
ulCmd	UINT32	0x00003B02	ETH_INTF_SEND_ETH_FRAME_REQ - Command
ulExt	UINT32	0	Extension. Set to zero.
ulRout	UINT32	0	Routing. Set to zero.
tData - Structure ETH_INTF_SEND_ETH_FRAME_REQ_DATA_T			
ulEthPort	UINT32	0 1 2	Use default port (According Switch Table) Send Ethernet Frame using Port 0 Send Ethernet Frame using Port 1
ulReserved	UINT32	0	Reserved for future usage. Set to zero
abData[1518]	UINT8		The content of the frame to send

Table 9: ETH_INTF_SEND_ETH_FRAME_REQ_T –Request Packet to send an Ethernet frame

3.1.3.2 Send Ethernet Frame Confirmation

Packet Structure Reference

```
/*****************************************************************************/
/*             Send Ethernet frame confirmation                              */
/*****************************************************************************/

typedef struct ETH_INTF_SEND_ETH_FRAME_CNF_PCK_Ttag ETH_INTF_SEND_ETH_FRAME_CNF_PCK_T;

struct  ETH_INTF_SEND_ETH_FRAME_CNF_PCK_Ttag
{
  /** packet header */
  TLR_PACKET_HEADER_T             tHead;
};
```

Packet Description

Structure ETH_INTF_SEND_ETH_FRAME_CNF_PCK_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32		Source End Point Identifier. Will contain same value as in Request.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Will contain same value as in Request.
ulSta	UINT32	0	Packet Status. Will be set to nonzero error code if service failed.
ulCmd	UINT32	0x00003B03	ETH_INTF_SEND_ETH_FRAME_CNF - command
ulExt	UINT32	0	Extension. Ignore.
ulRout	UINT32	0	Routing. Ignore

Table 10: ETH_INTF_SEND_ETH_FRAME_CNF_PCK_T –Confirmation Packet for Send Ethernet Frame Service

3.1.4 OEM Mode Received Ethernet Frame Service

This service is used by the Ethernet protocol task to indicate a frame received from Ethernet to host application. The precondition for this service is that the host application must have set a valid configuration, initialized the configuration and registered with the Ethernet protocol task. The host application is allowed to not send a response for this particular indication to save computing time.

3.1.4.1 Received Ethernet Frame Indication

Packet Structure Reference

```
/*****************************************************************************/
/*             Receive Ethernet frame indication                             */
/*****************************************************************************/
#define ETH_INTF_MAX_RECV_ETH_FRAME_SIZE  1518  /* Maximum data count for indication   */
#define ETH_INTF_RECV_ETH_FRAME_HEADER_SIZE  8  /* 8 Bytes not for header (port and
resrv)*/
#define ETH_INTF_RECV_ETH_FRAME_RSRV_SIZE    4  /* 4 Bytes reserved in indication     */

#define ETH_INTF_RECV_ETH_FRAME_PORT_0        0x00000001L
#define ETH_INTF_RECV_ETH_FRAME_PORT_1        0x00000002L

/** Receive Ethernet frame indication data */
typedef struct ETH_INTF_RECV_ETH_FRAME_IND_DATA_Ttag  ETH_INTF_RECV_ETH_FRAME_IND_DATA_T;

struct ETH_INTF_RECV_ETH_FRAME_IND_DATA_Ttag
{
  TLR_UINT32 ulEthPort;
  TLR_UINT32 ulReserved;

  TLR_UINT8  abData[ETH_INTF_MAX_RECV_ETH_FRAME_SIZE];
};

typedef struct ETH_INTF_RECV_ETH_FRAME_IND_PCK_Ttag ETH_INTF_RECV_ETH_FRAME_IND_PCK_T;

struct  ETH_INTF_RECV_ETH_FRAME_IND_PCK_Ttag
{
  /** Packet header */
  TLR_PACKET_HEADER_T                tHead;
  /** Packet data */
  ETH_INTF_RECV_ETH_FRAME_IND_DATA_T  tData;
};
```

Packet Description

Structure ETH_INTF_RECV_ETH_FRAME_IND_PCK_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32		Destination End Point Identifier. Will be set to the Source End Pointer Identifier of the Register Application Request
ulSrcId	UINT32		Source End Point Identifier. Will be set by Ethernet protocol task to an internal value. To be ignore by host application.
ulLen	UINT32	68 - 1526	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Will be Set to zero.
ulCmd	UINT32	0x00003B04	ETH_INTF_RECV_ETH_FRAME_IND - Command
ulExt	UINT32		Extension. Ignore.
ulRout	UINT32		Routing. Ignore.
tData - Structure ETH_INTF_RECV_ETH_FRAME_IND_DATA_T			
ulEthPort	UINT32	0 1 2	Reserved Ethernet Frame was received at Port 0 Ethernet Frame was received at Port 1
ulReserved	UINT32	0	Reserved for future usage. Ignore
abData[1518]	UINT8		The Content of the Received Frame

Table 11: ETH_INTF_RECV_ETH_FRAME_IND – Indication for receiving Ethernet frame

3.1.4.2 Received an Ethernet Frame Response

Packet Structure Reference

```
/*****************************************************************************/
/*             Receive Ethernet frame indication                             */
/*****************************************************************************/

/** Receive Ethernet frame response */
typedef struct ETH_INTF_RECV_ETH_FRAME_RES_PCK_Ttag ETH_INTF_RECV_ETH_FRAME_RES_PCK_T;

struct  ETH_INTF_RECV_ETH_FRAME_RES_PCK_Ttag
{
  /** packet header */
  TLR_PACKET_HEADER_T             tHead;
};
```

Packet Description

Structure ETH_INTF_RECV_ETH_FRAME_RES_PCK_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Use same value as in indication
ulSrc	UINT32		Source Queue-Handle. Use same value as in indication
ulDestId	UINT32		Destination End Point Identifier. Use same value as in indication
ulSrcId	UINT32		Source End Point Identifier. Use same value as in indication
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Use same value as in indication.
ulSta	UINT32	0	Packet Status. Ignored by Ethernet protocol task. Set to zero for future compatibility
ulCmd	UINT32	0x00003B05	ETH_INTF_RECV_ETH_FRAME_RES - Command
ulExt	UINT32	0	Extension. Use same value as in indication
ulRout	UINT32	0	Routing. Use same value as in indication

Table 12: ETH_INTF_RECV_ETH_FRAME_RES - Response for receiving Ethernet Frame

3.1.5 Ethernet Register Application Service

This outdated request is only available due to compatibility reasons. It has been replaced by the generic RCX_REGISTER_APP_REQ and RCX_UNREGISTER_APP_REQ services and shall not be used for future developments.

3.1.5.1 Ethernet Register Application Request

Packet Structure Reference

```
/*****************************************************************************/
/*             Register application request                                  */
/*****************************************************************************/
#define ETH_INTF_UNREGISTER_APPLICATION 0x00000000L
#define ETH_INTF_REGISTER_APPLICATION   0x00000001L

/** Register application request data */
typedef struct ETH_INTF_REGISTER_APP_REQ_DATA_Ttag  ETH_INTF_REGISTER_APP_REQ_DATA_T;

struct ETH_INTF_REGISTER_APP_REQ_DATA_Ttag
{
  TLR_UINT32 ulMode;
};

/** Register application request */
typedef struct ETH_INTF_REGISTER_APP_REQ_PCK_Ttag ETH_INTF_REGISTER_APP_REQ_PCK_T;

struct  ETH_INTF_REGISTER_APP_REQ_PCK_Ttag
{
  /** Packet header */
  TLR_PACKET_HEADER_T               tHead;
  /** Packet data */
  ETH_INTF_REGISTER_APP_REQ_DATA_T  tData;
};
```

Packet Description

Structure ETH_INTF_REGISTER_APP_REQ_PCK_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the ETH_INTF-Task process queue
ulSrc	UINT32	0	Source Queue-Handle. Set to zero
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32	arbitrary	Source End Point Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Unused in requests. Set to zero.
ulCmd	UINT32	0x00003B06	ETH_INTF_REGISTER_APP_REQ - Command
ulExt	UINT32	0	Extension. Set to zero.
ulRout	UINT32	0	Routing. Set to zero.
tData - Structure ETH_INTF_REGISTER_APP_REQ_DATA_T			
ulMode	UINT32	0 1	Unregister Application Register Application

Table 13: ETH_INTF_REGISTER_APP_REQ - Request to register/unregister application

3.1.5.2 Ethernet Register Application Confirmation

Packet Structure Reference

```
/*****************************************************************************/
/*             Register application confirmation                             */
/*****************************************************************************/

#define ETH_INTF_UNREGISTER_APPLICATION 0x00000000L
#define ETH_INTF_REGISTER_APPLICATION   0x00000001L

/** Register application comfirmation */
typedef struct ETH_INTF_REGISTER_APP_CNF_PCK_Ttag ETH_INTF_REGISTER_APP_CNF_PCK_T;

struct  ETH_INTF_REGISTER_APP_CNF_PCK_Ttag
{
  /** packet header */
  TLR_PACKET_HEADER_T             tHead;
};
```

Packet Description

Structure ETH_INTF_REGISTER_APP_CNF_PCK_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32		Source End Point Identifier. Will contain same value as in Request.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Will contain same value as in Request.
ulSta	UINT32	0	Packet Status. Will be set to nonzero error code if service failed.
ulCmd	UINT32	0x00003B07	ETH_INTF_REGISTER_APP_CNF - Command
ulExt	UINT32	0	Extension. Ignore.
ulRout	UINT32	0	Routing. Ignore

Table 14: ETH_INTF_REGISTER_APP_CNF - Confirmation of register/unregister application service

3.1.6 IP Configuration Service

This service is used by the Ethernet protocol task to provide details about the firmware's TCP/IP stack configuration. This request is issued on every configuration change of the TCP/IP stack. The precondition for this service is that the host application must have set a valid configuration, initialized the configuration and registered with the Ethernet protocol task.

3.1.6.1 IP Configuration Indication

Packet Structure Reference

```
/*****************************************************************************/
/*             Receive IP configuration indication                           */
/*****************************************************************************/

/* Flags ulFlags (from TcpipTcpTask_Public.h)! */
#define IP_CFG_FLAG_IP_ADDR             (0x0001)
#define IP_CFG_FLAG_NET_MASK            (0x0002)
#define IP_CFG_FLAG_GATEWAY             (0x0004)
#define IP_CFG_FLAG_BOOTP               (0x0008)
#define IP_CFG_FLAG_DHCP                (0x0010)
#define IP_CFG_FLAG_ETHERNET_ADDR       (0x0020)

/** Receive IP configuration indication */
typedef struct ETH_INTF_RECV_IP_CONFIG_IND_PCK_Ttag  ETH_INTF_RECV_IP_CONFIG_IND_PCK_T;

typedef struct ETH_INTF_RECV_IP_CONFIG_IND_DATA_Ttag
{
  TLR_UINT32  ulFlags;            /* Compatible to TCPIP_DATA_IP_CMD_SET_CONFIG_REQ_T */
  TLR_UINT32  ulIpAddr;           /* from TCP/IP stack                                */
  TLR_UINT32  ulNetMask;
  TLR_UINT32  ulGateway;
  TLR_UINT8   abEthernetAddr[6];

} ETH_INTF_RECV_IP_CONFIG_IND_DATA_T;

#define ETH_INTF_DATA_IP_CONFIG_FRAME_IND_SIZE  \
                (sizeof(ETH_INTF_RECV_IP_CONFIG_IND_DATA_T))

struct ETH_INTF_RECV_IP_CONFIG_IND_PCK_Ttag
{
  /** Packet header */
  TLR_PACKET_HEADER_T                tHead;
  /** Packet data */
  ETH_INTF_RECV_IP_CONFIG_IND_DATA_T  tData;
};
```

Packet Description

Structure ETH_INTF_RECV_IP_CONFIG_IND_PCK_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. ignore
ulSrc	UINT32		Source Queue-Handle. ignore
ulDestId	UINT32		Destination End Point Identifier. Will be set to the Source End Pointer Identifier of Register Application Request
ulSrcId	UINT32		Source End Point Identifier. Will be set by Ethernet protocol task to an internal value. To be ignore by host application.
ulLen	UINT32	22	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Will be Set to zero.
ulCmd	UINT32	0x00003B08	ETH_INTF_RECV_IP_CONFIG_IND - Command
ulExt	UINT32		Extension. Ignore.
ulRout	UINT32		Routing. Ignore.
tData - Structure ETH_INTF_RECV_IP_CONFIG_IND_DATA_T			
ulFlags	UINT32	0x0001 0x0002 0x0004 0x0008 0x0010 0x0020	IP Address value is valid and in use Network Mask value is valid and in use Gateway value is valid and in use BOOTP Protocol is enabled DHCP Protocol is enabled Ethernet MAC Address is Set
ulIpAddr	UINT32		The IP Address
ulNetMask	UINT32		The Network Mask
ulGateway	UINT32		The Gateway Address
abEthernetAddr[6]	UINT8		The Ethernet MAC Address

Table 15: ETH_INTF_RECV_IP_CONFIG_IND - Indication for IP configuration information

3.1.6.2 IP Configuration Response

Packet Structure Reference

```
/*****************************************************************************/
/*            Receive IP configuration response                              */
/*****************************************************************************/

/** Receive IP configuration response */
typedef struct ETH_INTF_RECV_IP_CONFIG_RES_PCK_Ttag  ETH_INTF_RECV_IP_CONFIG_RES_PCK_T;

struct  ETH_INTF_RECV_IP_CONFIG_RES_PCK_Ttag
{
  /** packet header */
  TLR_PACKET_HEADER_T            tHead;
};
```

Packet Description

Structure ETH_INTF_RECV_IP_CONFIG_RES_PCK_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Use same value as in indication
ulSrc	UINT32		Source Queue-Handle. Use same value as in indication
ulDestId	UINT32		Destination End Point Identifier. Use same value as in indication
ulSrcId	UINT32		Source End Point Identifier. Use same value as in indication
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Use same value as in indication.
ulSta	UINT32	0	Packet Status. Ignored by the Ethernet protocol task. Set to zero for future compatibility
ulCmd	UINT32	0x00003B09	ETH_INTF_RECV_ETH_FRAME_RES - Command
ulExt	UINT32	0	Extension. Use same value as in indication
ulRout	UINT32	0	Routing. Use same value as in indication

Table 16: ETH_INTF_RECV_IP_CONFIG_RES - Response for IP configuration information

3.1.7 Get Hardware Statistics Service

The Get Hardware Statistics Service can be used by the application to ready out several counters of the Ethernet switch. The availability of the different counters depends on the underlying Ethernet Switch Type.

3.1.7.1 Get Hardware Statistics Request

Packet Description

Structure ETH_INTF_GET_HW_STAT_PCK_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. ignore
ulSrc	UINT32		Source Queue-Handle. ignore
ulDestId	UINT32		Destination End Point Identifier. Will be set to the Source End Pointer Identifier of Register Application Request
ulSrcId	UINT32		Source End Point Identifier. Will be set by Ethernet protocol task to an internal value. To be ignore by host application.
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Will be set to zero.
ulCmd	UINT32	0x00003BFC	ETH_INTF_GET_HW_STAT_REQ - Command
ulExt	UINT32		Extension. Ignore.
ulRout	UINT32		Routing. Ignore.
tData - Structure ETH_INTF_GET_HW_STAT_REQ_DATA_T			
ulPortNo	UINT32	0,1	The port for which to retrieve the information

Table 17: ETH_INTF_GET_HW_STAT_REQ - Get hardware statistics request

3.1.7.2 Get Hardware Statistics Confirmation

Packet Description

Structure ETH_INTF_GET_HW_STAT_PCK_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Use same value as in indication
ulSrc	UINT32		Source Queue-Handle. Use same value as in indication
ulDestId	UINT32		Destination End Point Identifier. Use same value as in indication
ulSrcId	UINT32		Source End Point Identifier. Use same value as in indication
ulLen	UINT32	68	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Use same value as in indication.
ulSta	UINT32	0	Packet Status. Ignored by Ethernet protocol task. Set to zero for future compatibility
ulCmd	UINT32	0x00003BFD	ETH_INTF_GET_HW_STAT_CNF - Command
ulExt	UINT32	0	Extension. Use same value as in indication
ulRout	UINT32	0	Routing. Use same value as in indication
tData - Structure ETH_INTF_GET_HW_STAT_CNF_DATA_T			
ulPortNo	UINT32	0, 1	The port number. Taken unchanged from the request
ulTransmittedOkay	UINT32		Number of Tx Frames transmitted OK
ulSingleCollisionFrames	UINT32		Number of Tx Frames for which a single collision occurred
ulMultipleCollisionFrames	UINT32		Number of Tx Frames for which multiple collisions occurred
ulLateCollisions	UINT32		Number of Tx Frames for which a late collusion occurred
ulLinkdownDuringTransmission	UINT32		Number of Tx Frames for which a link down occurred.
ulUtxUnderflowDuringTransmission	UINT32		
ulFramesReceivedOkay	UINT32		Number of Rx Frames received OK.
ulFCSErrors	UINT32		Number of Rx Frames for which a FCS error was detected
ulAlignmentErrors	UINT32		Number of Rx Frames for which an alignment error occurred
ulFrameTooLongErrors	UINT32		Number of Rx Frames which exceeded the maximum Ethernet Frame length
ulRuntFrames	UINT32		
ulCollisionFragmentsReceived	UINT32		Number of Rx Frames for which a collision was signalled.
ulDroppedDueLowResource	UINT32		Number of Frames dropped in switch because of no memory available
ulUrxOverflows	UINT32		
ulRxFifoFill	UINT32		Current Fill Level of receive FIFO
ulTxFifoFill	UINT32		Current Fill Level of transmit FIFO

Table 18: ETH_INTF_GET_HW_STAT_CNF - Get Hardware Statistics Confirmation

3.2 Ethernet Protocol API in Ethernet (NDIS) Mode

The following packets are available when using the Ethernet Protocol API in Ethernet (NDIS) Mode:

Overview over the Packets of the Ethernet Stack (OEM Mode)			
Section	**Packet**	**Command code**	**Page**
3.2.2	Event Indication	0x3B20	34
	Event Response	0x3B21	35
3.2.3	Send Ethernet Frame Request	0x3B22	36
	Send Ethernet Frame Confirmation	0x3B23	37
3.2.4	Received Ethernet Frame Indication	0x3B24	38
	Received an Ethernet Frame Response	0x3B25	39
3.2.5	Set Multicast Single Request	0x3B26	40
	Set Multicast Single Confirmation	0x3B27	41
3.2.6	Clear Multicast Single Request	0x3B28	42
	Clear Multicast Single Confirmation	0x3B29	43

Table 19: Overview over the Packets of the Ethernet Protocol Stack (NDIS Mode)

Note: Availability of this mode depends on the Industrial Ethernet stack!

In this operation mode, the Ethernet protocol task does not need to be configured.

The following topics have to be taken into account when using the Ethernet Protocol API in Ethernet (NDIS) Mode:

[1] In order to receive any indications the host application shall register itself using the generic `RCX_REGISTER_APP_REQ` service.

[2] The host application will be visible to the network with its dedicated MAC Address appearing as individual device, i.e. it uses a different MAC Address than the Industrial Ethernet protocol running within the netX.

[3] As any Ethernet Protocol parameter can be chosen freely with respect to the Protocol Standard and Definitions, parameter conflicts must be avoided by the host application.

[4] Therefore, never use the same IP address as the Industrial Ethernet protocol.

[5] The Ethernet Switch might filter certain Ethernet Frames sent or received by the host application (depending on protocol/firmware, see API manual).

3.2.1 Common Services

The Ethernet protocol task supports the following common services which are described in *Dual-Port Memory Interface Manual for netX based Products* [1]:

- **Register Application Service**: used to register application.
- **Unregister Application Service**: used to unregister application.
- **Identify Firmware Service**: used to retrieve identification information from Ethernet protocol task.

3.2.2 Event Service

The Event Service is used by the Ethernet protocol task to notify the host application about state changes. As a prerequisite the host application must register with Ethernet protocol task. A simple locking mechanism is used to prevent flooding the host application with Event Indications. Thus, the host application must send a valid Event Response for each received Event Indication. If the host application fails in this, no further Event Indications will be sent by the Ethernet protocol task.

3.2.2.1 Event Indication

Packet Description

Structure `ETH_INTF_EVENT_IND_T`			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure `TLR_PACKET_HEADER_T`			
ulDest	UINT32		Destination Queue-Handle. ignore
ulSrc	UINT32		Source Queue-Handle. ignore
ulDestId	UINT32		Destination End Point Identifier. Will be set to the Source End Pointer Identifier of Register Application Request
ulSrcId	UINT32		Source End Point Identifier. Will be set by Ethernet protocol task to an internal value. To be ignore by host application.
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Will be Set to zero.
ulCmd	UINT32	0x00003B20	`ETH_INTF_CMD_EVENT_IND` - Command
ulExt	UINT32		Extension. Ignore.
ulRout	UINT32		Routing. Ignore.
tData - Structure `ETH_INTF_RECV_IP_CONFIG_IND_DATA_T`			
uiEventCnt[2]	UINT16		Array of Event Counters. Each counter contains the number of corresponding events occurred since last Event Indication had been sent

Table 20: `ETH_INTF_CMD_EVENT_IND` - Indication of events

Currently the following events are defined

Constant	Event Id	Description
ETH_INTF_EVENT_LINKCHANGED	0	Link Status has Changed. Current Link Status may be derived from Extended Status Block
ETH_INTF_EVENT_IPCHANGED	1	netX internal TCP/IP stack configuration had changed. Current settings may be derived from Extended Status Block.

Table 21: Event Descriptions

3.2.2.2 Event Response

Packet Description

Structure ETH_INTF_EVENT_RSP_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Use same value as in indication
ulSrc	UINT32		Source Queue-Handle. Use same value as in indication
ulDestId	UINT32		Destination End Point Identifier. Use same value as in indication
ulSrcId	UINT32		Source End Point Identifier. Use same value as in indication
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Use same value as in indication.
ulSta	UINT32	0	Packet Status. Ignored by Ethernet protocol task. Set to zero for future compatibility
ulCmd	UINT32	0x00003B21	ETH_INTF_RECV_ETH_FRAME_RES - Command
ulExt	UINT32	0	Extension. Use same value as in indication
ulRout	UINT32	0	Routing. Use same value as in indication

Table 22: ETH_INTF_CMD_EVENT_RSP – Response to event indication

3.2.3 Ethernet (NDIS) Mode Send Ethernet Frame Service

This service shall be used to send a frame to Ethernet. The minimum length of an Ethernet Frame is 60 bytes and the Maximum is 1518 Bytes.

3.2.3.1 Send Ethernet Frame Request

Packet Description

Structure ETH_INTF_SEND_ETH_FRAME_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the ETH_INTF-Task process queue
ulSrc	UINT32	0	Source Queue-Handle. Set to zero
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32	arbitrary	Source End Point Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulLen	UINT32	60 - 1518	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Unused in requests. Set to zero.
ulCmd	UINT32	0x00003B22	ETH_INTF_CMD_SEND_ETH_FRAME_REQ - Command
ulExt	UINT32	0	Extension, Set to zero.
ulRout	UINT32	0	Routing. Set to zero.
tData - Structure ETH_INTF_ETH_FRAME_T			
abDestMacAddr[6]	UINT8		Ethernet Destination MAC Address
abSrcMacAddr[6]	UINT8		The source MAC Address. This field can be left empty. It will be forced by the Ethernet Switch to the correct value. (Fourth netX MAC Adress)
abData[1506]	UINT8		Data Payload of Ethernet Frame (starting from ether type field)

Table 23: ETH_INTF_CMD_SEND_ETH_FRAME_REQ – Request to send an Ethernet frame

3.2.3.2 Send Ethernet Frame Confirmation

Packet Description

Structure ETH_INTF_SEND_ETH_FRAME_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32		Source End Point Identifier. Will contain same value as in Request.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Will contain same value as in Request.
ulSta	UINT32	0	Packet Status. Will be set to nonzero error code if service failed.
ulCmd	UINT32	0x00003B23	ETH_INTF_CMD_SEND_ETH_FRAME_CNF - Command
ulExt	UINT32	0	Extension. Ignore.
ulRout	UINT32	0	Routing. Ignore

Table 24: ETH_INTF_CMD_SEND_ETH_FRAME_CNF – Confirmation for Send Ethernet Frame Service

3.2.4 Ethernet (NDIS) Mode Received Ethernet Frame Service

This service is used by the Ethernet protocol task to indicate a frame received from Ethernet to host application. As a prerequisite the host application must register with Ethernet protocol task. The host application is allowed to not send a response for this particular indication to save computing time.

3.2.4.1 Received Ethernet Frame Indication

Packet Description

Structure ETH_INTF_RECV_ETH_FRAME_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. ignore
ulSrc	UINT32		Source Queue-Handle. ignore
ulDestId	UINT32		Destination End Point Identifier. Will be set to the Source End Pointer Identifier of Register Application Request
ulSrcId	UINT32		Source End Point Identifier. Will be set by Ethernet protocol task to an internal value. To be ignore by host application.
ulLen	UINT32	60 - 1518	Packet Data Length in bytes
ulId	UINT32	Arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Will be set to zero.
ulCmd	UINT32	0x00003B24	ETH_INTF_CMD_RECV_ETH_FRAME_IND - Command
ulExt	UINT32		Extension. Ignore.
ulRout	UINT32		Routing. Ignore.
tData - Structure ETH_INTF_ETH_FRAME_T			
abDestMacAddr[6]	UINT8		Ethernet Destination MAC Address
abSrcMacAddr[6]	UINT8		The source MAC Address.
abData[1506]	UINT8		Data Payload of Ethernet Frame (starting from ether type field)

Table 25: ETH_INTF_CMD_RECV_ETH_FRAME_IND – Indication for receiving Ethernet frame

3.2.4.2 Received an Ethernet Frame Response

Packet Description

Structure ETH_INTF_RECV_ETH_FRAME_RSP_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Use same value as in Indication
ulSrc	UINT32		Source Queue-Handle. Use same value as in Indication
ulDestId	UINT32		Destination End Point Identifier. Use same value as in Indication
ulSrcId	UINT32		Source End Point Identifier. Use same value as in Indication
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Use same value as in Indication.
ulSta	UINT32	0	Packet Status. Ignored by Ethernet Protocol Task. Set to zero for future compatibility
ulCmd	UINT32	0x00003B25	ETH_INTF_CMD_RECV_ETH_FRAME_RSP - Command
ulExt	UINT32	0	Extension. Use same value as in Indication
ulRout	UINT32	0	Routing. Use same value as in Indication

Table 26: ETH_INTF_CMD_RECV_ETH_FRAME_RSP – Indication for receiving Ethernet frame

3.2.5 Ethernet (NDIS) Mode Set Multicast Single Service

This service shall be used to receive traffic from a specific IPv4 multicast group, i.e. to enable frame reception which target the given destination multicast MAC address. A netX device can participate in a minimum of 32 multicast groups. Once set, frames targeting the specified destination MAC address are forwarded by means of the Received Ethernet Frame Service as described in 3.1.4.1.

This service is usually only supported by Firmwares which actually implement a Hardware-Level Multicast Filter. Firmwares that don't implement such a filter usually forward all received multicast traffic towards the Host.

Since the actual implementation of Multicast Filtering may make use of non-collision-free hashing, Multicast traffic for other group addresses with the same hash value also may pass the filter once a given group address is set via this service.

Depending on the Industrial Protocol Stack used, there may be further restrictions on certain frames not to be forwarded towards the Ethernet Interface in order to avoid disturbing realtime data exchange in the Industrial Ethernet Protocol.

Note: This service is available since V4.4.0.0 of Ethernet Interface component.
Availability of this feature depends on the Industrial Ethernet stack!

3.2.5.1 Set Multicast Single Request

Packet Description

Structure `ETH_INTF_SET_MULTICAST_SINGLE_REQ_T`			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure `TLR_PACKET_HEADER_T`			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the ETH_INTF-Task process queue
ulSrc	UINT32	0	Source Queue-Handle. Set to zero
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32	arbitrary	Source End Point Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulLen	UINT32	6	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Unused in requests. Set to zero.
ulCmd	UINT32	0x00003B26	`ETH_INTF_SET_MULTICAST_SINGLE_REQ` - Command
ulExt	UINT32	0	Extension, Set to zero.
ulRout	UINT32	0	Routing. Set to zero.
tData – Structure			
abMacAddr[6]	UINT8		Ethernet Multicast Group Address

Table 27: `ETH_INTF_SET_MULTICAST_SINGLE_REQ` – Request to receive from a specific multicast group

3.2.5.2 Set Multicast Single Confirmation

Packet Description

Structure ETH_INTF_SET_MULTICAST_SINGLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32		Source End Point Identifier. Will contain same value as in Request.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Will contain same value as in Request.
ulSta	UINT32	0	Packet Status. Will be set to nonzero error code if service failed.
ulCmd	UINT32	0x00003B27	ETH_INTF_CMD_SET_MULTICAST_SINGLE_CNF - Command
ulExt	UINT32	0	Extension. Ignore.
ulRout	UINT32	0	Routing. Ignore

Table 28: ETH_INTF_SET_MULTICAST_SINGLE_CNF – Confirmation to receive from a specific multicast group

3.2.6 Ethernet (NDIS) Mode Clear Multicast Single Service

This service shall be used to stop receiving traffic from a specific IPv4 multicast group, i.e. to disable frame reception which target the given destination multicast MAC address. A netX device can participate in a minimum of 32 multicast groups. Once cleared, frames targeting the specified destination MAC address which has previously been set with the Set Multicast Single Service (as described in section 3.2.5) are no longer forwarded by means of the Received Ethernet Frame Service as described in 3.1.4.1.

This service is usually only supported by Firmwares which actually implement a Hardware-Level Multicast Filter. Firmwares that don't implement such a filter usually forward all received multicast traffic towards the Host.

Since the actual implementation of Multicast Filtering may make use of non-collision-free hashing, Multicast traffic for other group addresses with the same hash value also may stop passing the filter once a given group address is cleared via this service.

Note: This service is available since V4.4.0.0 of Ethernet Interface component.

Availability of this feature depends on the Industrial Ethernet stack!

3.2.6.1 Clear Multicast Single Request

Packet Description

Structure `ETH_INTF_CLR_MULTICAST_SINGLE_REQ_T`			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure `TLR_PACKET_HEADER_T`			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the ETH_INTF-Task process queue
ulSrc	UINT32	0	Source Queue-Handle. Set to zero
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32	arbitrary	Source End Point Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulLen	UINT32	6	Packet Data Length in bytes
ulId	UINT32	arbitrary	Packet Identifier. This field can be used by host application for internal purposes. Will be reused in confirmation.
ulSta	UINT32	0	Packet Status. Unused in requests. Set to zero.
ulCmd	UINT32	0x00003B28	`ETH_INTF_CLR_MULTICAST_SINGLE_REQ` - Command
ulExt	UINT32	0	Extension, Set to zero.
ulRout	UINT32	0	Routing. Set to zero.
tData – Structure			
abMacAddr[6]	UINT8		Ethernet Multicast Group Address

Table 29: `ETH_INTF_CLR_MULTICAST_SINGLE_REQ` – Request to not receive from a specific multicast group

3.2.6.2 Clear Multicast Single Confirmation

Packet Description

Structure ETH_INTF_CLR_MULTICAST_SINGLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x00000020	Destination Queue-Handle. Ignore.
ulSrc	UINT32		Source Queue-Handle. Ignore
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to zero.
ulSrcId	UINT32		Source End Point Identifier. Will contain same value as in Request.
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32		Packet Identifier. Will contain same value as in Request.
ulSta	UINT32	0	Packet Status. Will be set to nonzero error code if service failed.
ulCmd	UINT32	0x00003B29	ETH_INTF_CMD_CLR_MULTICAST_SINGLE_CNF - Command
ulExt	UINT32	0	Extension. Ignore.
ulRout	UINT32	0	Routing. Ignore

Table 30: ETH_INTF_CLR_MULTICAST_SINGLE_CNF – Confirmation to not receive from a specific multicast group

3.3 TCP/UDP Protocol API

The TCP/UDP interface is common to both operational modes. The interface to netX internal TCP/IP Task is implemented by routing the packets between the host application and the internal TCP/IP stack. A description of the API is given by the TCP/IP Protocol API Manual. Please refer to this document for detailed information.

In order to prevent any impact on the Industrial Ethernet protocol, the Ethernet protocol task might filter dangerous requests. This setting depends on the Firmware and is specified within the corresponding firmware/protocol Manual. The Set Configuration confirmation will contain the value `ETH_INTF_SET_CONFIG_CNF_TCPIP_STATUS_RESTRICTED` in field `ulStatusTcpIp`, if filtering is active. When a request is rejected by Ethernet protocol task the request packet will be responded using the generic code `TLR_E_FAIL`. Any command in the range from 0x00000200 to 0x000002FF (Command definition starting with `TCPIP_IP_CMD_`) is affected from this filtering. In the current implementation the following requests are still allowed in restricted scenarios:

- TCPIP_IP_CMD_GET_CONFIG_REQ
- TCPIP_IP_CMD_GET_PARAM_REQ
- TCPIP_IP_CMD_GET_OPTIONS_REQ
- TCPIP_IP_CMD_PING_REQ
- TCPIP_IP_CMD_SET_PARAM_REQ if used in conjunction with ulMode value of
  - IP_PRM_SEND_ARP_REQ
  - IP_PRM_SEND_ARP_TMT_REQ

Besides this filtering it might be possible that some IP ports are used by the Industrial Ethernet protocol and thus are not available for use by the host application. Details about these restrictions are described in detail in the corresponding protocol/firmware manual.

Depending on the operational mode it may be required to configure the Ethernet Interface task. In mode `RCX_PROT_CLASS_OEM` it is required to first configure the task with Set Configuration Service. Here `ulModeFlags` needs to be set to 0x00000002 (Enable TCP/UDP Interface). Afterwards this configuration needs to be activated by using Channel Initialization Service. Now it is possible to use the TCP/UDP Protocol API.

In mode it is not required to configure the task upfront.

4 Status/Error Codes

4.1 Packet Status/Error

The following status and error codes are used by the Ethernet protocol task in the `ulSta` field of confirmation packets:

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC05D0001	TLR_E_ETH_INTF_COMMAND_INVALID Invalid command received.
0xC05D0002	TLR_E_ETH_INTF_CONFIG_LOCK Configuration is locked.
0xC05D0003	TLR_E_ETH_INTF_INVALID_PACKET_LENGHT Invalid packet length.
0xC05D0004	TLR_E_ETH_INTF_INVALID_MODE Invalid mode in request.
0xC05D0005	TLR_E_ETH_INTF_PARAM_AUTO_NEGOTIATION_PORT_0 Invalid parameter for auto-negotiation port 0.
0xC05D0006	TLR_E_ETH_INTF_PARAM_AUTO_NEGOTIATION_PORT_1 Invalid parameter for auto-negotiation port 1.
0xC05D0007	TLR_E_ETH_INTF_PARAM_DUPLEX_MODE_PORT_0 Invalid parameter for duplex mode port 0.
0xC05D0008	TLR_E_ETH_INTF_PARAM_DUPLEX_MODE_PORT_1 Invalid parameter for duplex mode port 1.
0xC05D0009	TLR_E_ETH_INTF_PARAM_TRANSMISSION_RATE_PORT_0 Invalid parameter for transmission rate port 0.
0xC05D000A	TLR_E_ETH_INTF_PARAM_TRANSMISSION_RATE_PORT_1 Invalid parameter for transmission rate port 1.
0xC05D000B	TLR_E_ETH_INTF_PARAM_AUTO_CROSSOVER_PORT_0 Invalid parameter for auto cross-over port 0.
0xC05D000C	TLR_E_ETH_INTF_PARAM_AUTO_CROSSOVER_PORT_1 Invalid parameter for auto cross-over port 1.
0xC05D000D	TLR_E_ETH_INTF_NO_CONFIGURATION Task is not configured.
0xC05D000E	TLR_E_ETH_INTF_APP_NOT_REGISTERED No application registered.
0xC05D000F	TLR_E_ETH_INTF_APP_SET_NOT_READY Application set not ready.
0xC05D0010	TLR_E_ETH_INTF_LINK_DOWN No Ethernet link.
0xC05D0011	TLR_E_ETH_INTF_GET_SEND_BUFFER Failed to get send buffer.
0xC05D0012	TLR_E_ETH_INTF_SEND_FRAME Failed to send Ethernet frame.
0xC05D0013	TLR_E_ETH_INTF_SET_DRV_EDD_CFG Failed to set driver EDD configuration.
0xC05D0014	TLR_E_ETH_INTF_INVALID_ETH_PORT Invalid parameter for Ethernet port.
0xC05DFFFF	TLR_E_ETH_INTF_UNKNOWN_ERROR Unknown error detected.
	TLR_E_UNEXPECTED / TLR_E_FAIL Raw edd interface / TCP/UDP interface not available

Table 31: Packet Status/Error Codes Overview

4.2 Diagnostic Status/Error

The following status and error codes can be read by the application from the extended diagnostic block in the dual-port memory.

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok.
0xC05D0001	TLR_DIAG_E_ETH_INTF_INVALID_STARTUP_PARAMETER Invalid start-up parameter.
0x405D0002	TLR_DIAG_I_ETH_INTF_UNKNOWN_COMMAND_RECEIVED Unknown command received.
0xC05D0003	TLR_DIAG_E_ETH_INTF_SEND_PACKET_FAILED Failed to send packet.
0xC05D0004	TLR_DIAG_E_ETH_INTF_GET_PACKET_FAILED Failed to get packet.
0xC05D0005	TLR_DIAG_E_ETH_INTF_PACKET_DONE_FAILED Failed to return or release packet.
0xC05D0006	TLR_DIAG_E_ETH_INTF_SET_DRV_EDD_CFG Failed to set driver EDD configuration.
0xC05D0007	TLR_DIAG_E_ETH_INTF_GET_BUFFER Failed to get Ethernet buffer.
0xC05D0008	TLR_DIAG_E_ETH_INTF_FREE_BUFFER Failed to free Ethernet buffer.
0xC05D0009	TLR_DIAG_E_ETH_INTF_SEND_BUFFER Failed to send Ethernet buffer.
0xC05D000A	TLR_DIAG_E_ETH_INTF_GET_DRV_EDD_DIAG Failed to get diagnostic information from driver EDD.
0xC05D000B	TLR_DIAG_E_ETH_INTF_INVALID_ETH_PORT Invalid parameter for Ethernet port.

Table 32: Diagnostic Status/Error Codes Overview

5 Appendix

5.1 Extended Status

A communication channel represents a part of the Dual Port Memory. As explained in the *Dual-Port Memory Interface Manual for netX based Products* [1], it usually consists of the following elements:

- Output Data Image - is used to transfer cyclic process data to the network (normal or high-priority)
- Input Data Image - is used to transfer cyclic process data from the network (normal or high-priority)
- Send Mailbox - is used to transfer non-cyclic data to the netX
- Receive Mailbox - is used to transfer non-cyclic data from the netX
- Control Block - allows the host system to control certain channel functions
- Common Status Block - holds information common to all protocol stacks
- Extended Status Block - holds protocol specific network status information

This section explains the meaning of the only protocol-specific element of these, namely the Extended Status Block, within the context of the Ethernet Protocol Stack.

Note: All offsets mentioned in this section are relative to the beginning of the common status block, as the start offset of this block depends on the size and location of the preceding blocks.

```
typedef struct tagNETX_EXTENDED_STATUS_BLOCK
{
UINT8 abExtendedStatus[432];
} NETX_EXTENDED_STATUS_BLOC
```

Depending on the operational mode, the Ethernet protocol task provides different information within the extended status block.

If OEM mode is operational, the Extended Status Block update is synchronized with the DPM input area handshake. In order to update the content of extended status block this area should be written. (Typically using zero length data)

The following table describes the content of the Extended Status Block, if OEM mode is operational:

Information	Type	Description
`abEthernetAddrPort0[6]`	UINT8	The Ethernet MAC Address assigned to the Ethernet Interface Port 0
`abEthernetAddrPort1[6]`	UINT8	The Ethernet MAC Address assigned to the Ethernet Interface Port 1
`ulLinkStatusPort0`	UINT32	Nonzero if Link at Port 0 established
`ulLinkStatusPort1`	UINT32	Nonzero if Link at Port 1 established
`ulAutoNegotiationPort0`	UINT32	Nonzero if Ethernet Link Autonegotiation enabled for Ethernet Interface Port 0
`ulAutoNegotiationPort1`	UINT32	Nonzero if Ethernet Link Autonegotiation enabled for Ethernet Interface Port 1
`ulDuplexMode0`	UINT32	Nonzero if Full Duplex Link at Ethernet Interface Port 0 detected. Zero on Half Duplex Link at Ethernet Interface Port 0

Information	Type	Description
ulDuplexMode1	UINT32	Nonzero if Full Duplex Link at Ethernet Interface Port 1 detected. Zero on Half Duplex Link at Ethernet Interface Port 1
ulReserved1Port0	UINT32	Reserved. Do not evaluate.
ulReserved1Port1	UINT32	Reserved. Do not evaluate.
ulReserved2Port0	UINT32	Reserved. Do not evaluate.
ulReserved2Port1	UINT32	Reserved. Do not evaluate.
ulRxCntPort0	UINT32	Counter for frames received at Ethernet Interface Port 0
ulRxCntPort1	UINT32	Counter for frames received at Ethernet Interface Port 1
ulTxCntPort0	UINT32	Counter for frames sent at Ethernet Interface Port 0
ulTxCntPort1	UINT32	Counter for frames sent at Ethernet Interface Port 1
ulRxErrCntPort0	UINT32	Counter for frame receive errors at Ethernet Interface Port 0
ulRxErrCntPort1	UINT32	Counter for frame receive errors at Ethernet Interface Port 1
ulTxErrCntPort0	UINT32	Counter for frame transmit errors at Ethernet Interface Port 0
ulTxErrCntPort1	UINT32	Counter for frame transmit errors at Ethernet Interface Port 1
ulRxEthOverrunCntPort0	UINT32	Counter for Ethernet Interface Port 0 MAC receive buffer overruns
ulRxEthOverrunCntPort1	UINT32	Counter for Ethernet Interface Port 1 MAC receive buffer overruns
ulTxEthOverrunCntPort0	UINT32	Counter for Ethernet Interface Port 0 MAC transmit buffer overruns
ulTxEthOverrunCntPort1	UINT32	Counter for Ethernet Interface Port 1 MAC transmit buffer overruns
ulRxQueOverrunCntPort0	UINT32	Counter for receive queue overrun errors at Ethernet Interface Port 0
ulRxQueOverrunCntPort1	UINT32	Counter for receive queue overrun errors at Ethernet Interface Port 1
ulTxQueOverrunCntPort0	UINT32	Counter for send queue overrun errors at Ethernet Interface Port 0
ulTxQueOverrunCntPort1	UINT32	Counter for send queue overrun errors at Ethernet Interface Port 1
ulRxQueFillLevelPort0	UINT32	Fill level of receive queue for Ethernet Interface Port 0
ulRxQueFillLevelPort1	UINT32	Fill level of receive queue for Ethernet Interface Port 1
ulTxQueFillLevelPort0	UINT32	Fill level of transmit queue for Ethernet Interface Port 0
ulTxQueFillLevelPort1	UINT32	Fill level of transmit queue for Ethernet Interface Port 1
ulEthIntfTskFlags	UINT32	Ethernet Interface Task Flags. See Table 33
ulEthIntfTskStatusUpdateCnt	UINT32	Counter for Extend Status Area Updates
ulEthIntfTskError	UINT32	Nonzero error code if error condition is pending
ulEthIntfTskErrCnt	UINT32	Counter for error events

Table 33: Diagnostic Information in OEM operation mode

Constant	Value	Description
ETH_INTF_TSK_FLAGS_CONFIGURED	0x00000001	The task had been successfully configured.
ETH_INTF_TSK_FLAGS_APP_REGISTER	0x00000002	An host application had been registered with task
ETH_INTF_TSK_FLAGS_APP_NOT_RDY	0x00000004	The host application did not set the ready flag.
ETH_INTF_TSK_FLAGS_APP_CFG_LOCK	0x00000008	The host application set configuration lock.
ETH_INTF_TSK_FLAG_COM_READY	0x00000100	The Ethernet Protocol is ready for communication
ETH_INTF_TSK_FLAGS_LINK_0_OK	0x00010000	Ethernet Interface Port 0 has a link
ETH_INTF_TSK_FLAGS_LINK_1_OK	0x00020000	Ethernet Interface Port 1 has a link

Table 34: Meaning of individual Ethernet Interface Task Flag Bits

If Ethernet (NDIS) mode is operational, the content of Extended Status Block is defined as follows:

Information	Type	Description
abMacAddress[6]	UINT8	The Ethernet MAC Address assigned to the Ethernet Interface
bMautype	UINT8	The current MAU type (connection type) of the Interface. For specific Values, see IANA MauType Mib.
bPadding	UINT8	Padding for Alignment.
ulIpAddress	UINT32	The IP Address of the firmware's internal TCP/IP Stack
ulNetMask	UINT32	The Network Mask of the firmware's internal TCP/IP Stack
ulGateway	UINT32	The Gateway address of the firmware's internal TCP/IP Stack
ulIfInPkts	UINT64	Number of received Ethernet frames transferred to DPM
ulIfInDiscards	UINT64	Number of received Ethernet frames dropped because of missing resources
ulIfOutPkts	UINT64	Number of sent Ethernet frames
ulIfOutDiscards	UINT64	Numer of Ethernet frames not sent because of missing resources
ulIfInBytes	UINT64	Counter of bytes received at host application.
ulIfOutBytes	UINT64	Counter of bytes sent by the host application

Table 35: Diagnostic Information in Ethernet (NDIS) operation mode

Note: The structures are also defined within the header file EthIntf_Public.h.

5.2 List of Tables

5.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com
Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

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

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

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

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

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

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

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch
Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

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