



**Protocol API**  
**CC-Link Slave**

V2.4.x.x

**Hilscher Gesellschaft für Systemautomation mbH**

**[www.hilscher.com](http://www.hilscher.com)**

DOC071101API06EN | Revision 6 | English | 2010-06 | Released | Public

## Revision History

Rev	Date	Name	Revisions
1	20.11.07	ES	Created. CC-Link slave firmware version 2.0.1
2	25.01.08	ES	Support of CC-Link Version 2.0. Firmware/stack version 2.1.4
3	06.05.08	RG	Unified description of WS parameters and IO data. Changed notation of WS parameters. Restructuring.
4	26.11.08	RG	Firmware/ stack version V2.2.2 Reference to netX Dual-Port Memory Interface Manual Revision 7. Warmstart -> Set Configuration Registration/unregistration packet marked as obsolete. Changed some error numbers to global error numbers Added section on task structure.
5	27.08.09	RG	Firmware/ stack version V2.2.2 Added 3 error message descriptions Extended section on task structure with descriptions of single tasks. Added information of suitability of packets for LFW or LOM approach. Error corrections in text and task structure graphics ( <i>Table 14: Communication State of Change</i> ).
6	15.03.10	RG	Firmware/ stack version V2.4.x Parameter <code>ulIoTypesPoints</code> added Legal texts added Footer adapted Section <i>Technical Data</i> : New: Support of DMA for PCI targets

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>8</b>
1.1	Abstract .....	8
1.2	System Requirements .....	8
1.3	Intended Audience .....	8
1.4	Specifications .....	9
1.4.1	Protocol Task System .....	9
1.4.2	Technical Data .....	9
1.5	Terms, Abbreviations and Definitions .....	11
1.6	References .....	11
1.7	Legal Notes .....	12
1.7.1	Copyright.....	12
1.7.2	Important Notes.....	12
1.7.3	Exclusion of Liability.....	13
1.7.4	Export.....	13
<b>2</b>	<b>Fundamentals .....</b>	<b>14</b>
2.1	General Access Mechanisms on netX Systems .....	14
2.2	Accessing the Protocol Stack by Programming the AP Task's Queue .....	15
2.2.1	Getting the Receiver Task Handle of the Process Queue.....	15
2.2.2	Meaning of Source- and Destination-related Parameters .....	15
2.3	Accessing the Protocol Stack via the Dual Port Memory Interface.....	16
2.3.1	Communication via Mailboxes.....	16
2.3.2	Using Source and Destination Variables correctly .....	17
2.3.3	Obtaining useful Information about the Communication Channel .....	20
2.4	Client/Server Mechanism .....	22
2.4.1	Application as Client.....	22
2.4.2	Application as Server .....	23
<b>3</b>	<b>Dual-Port Memory .....</b>	<b>24</b>
3.1	Cyclic Data (Input/Output Data) .....	24
3.1.1	Input Process Data.....	25
3.1.2	Output Process Data.....	25
3.2	Acyclic Data (Mailboxes).....	26
3.2.1	General Structure of Messages or Packets for Non-Cyclic Data Exchange.....	27
3.2.2	Status & Error Codes .....	30
3.2.3	Differences between System and Channel Mailboxes .....	30
3.2.4	Send Mailbox .....	31
3.2.5	Receive Mailbox.....	31
3.2.6	Channel Mailboxes (Details of Send and Receive Mailboxes) .....	31
3.3	Status .....	32
3.3.1	Common Status .....	32
3.3.2	Extended Status.....	39
3.4	Control Block .....	51
<b>4</b>	<b>Getting started / Configuration .....</b>	<b>52</b>
4.1	Overview about Essential Functionality .....	52
4.2	Warmstart Parameters .....	53
4.2.1	Behavior when receiving a Set Configuration / Warmstart Command.....	55
4.3	Input and Output Data .....	56
4.3.1	Input and Output Data for CC-Link Version 1.....	56
4.3.2	Input and Output Data for CC-Link Version 2.....	58
4.4	Task Structure of the CC-Link Slave Stack.....	62
<b>5</b>	<b>The Application Interface.....</b>	<b>64</b>
5.1	The CC-Link APS-Task.....	65
5.1.1	CCLINK_APS_WARMSTART_REQ/CNF – Set Warmstart Parameters.....	66
5.1.2	CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration .....	76
5.2	The CC-Link Slave-Task .....	82
5.2.1	CCLINK_SLAVE_INITIALIZE_REQ/CNF – Initialization of CC-Link Slave .....	84
5.2.2	CCLINK_SLAVE_REGISTER_REQ/CNF – Register Application .....	87
5.2.3	CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle.....	91

---

5.2.4	CCLINK_SLAVE_SET_BUSPARAM_REQ/CNF – Set Bus Parameters .....	95
5.2.5	CCLINK_SLAVE_STARTSTOP_REQ/CNF – Start/Stop Communication with Network.....	101
5.2.6	CCLINK_SLAVE_GET_CCL_STATUS_REQ/CNF – Get CC-Link Status .....	105
5.2.7	CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ/CNF – Change CC-Link Slave Status .....	109
5.2.8	CCLINK_SLAVE_GET_BUS_PARAM_REQ/CNF – Get Bus Parameters.....	115
5.2.9	CCLINK_SLAVE_STATE_CHANGE_IND/RES – Change of State Indication .....	118
5.2.10	CCLINK_SLAVE_SET_WATCHDOG_FAIL_REQ/CNF – Set Watchdog Fail.....	123
<b>6</b>	<b>Status/Error Codes Overview .....</b>	<b>125</b>
6.1	Status/Error Codes CC-Link APS-Task .....	125
6.2	Status/Error codes CC-Link Slave-Task .....	127
<b>7</b>	<b>Contact.....</b>	<b>129</b>

---

## List of Figures

Figure 1 - The three different Ways to access a Protocol Stack running on a netX System .....	14
Figure 2 - Use of <code>ulDest</code> in Channel and System Mailbox .....	17
Figure 3 - Using <code>ulSrc</code> and <code>ulSrcId</code> .....	18
Figure 4: Transition Chart Application as Client.....	22
Figure 5: Transition Chart Application as Server.....	23
Figure 6: Internal Structure of CC-Link Slave Firmware.....	62

## List of Tables

Table 1: Names of Tasks in CC-Link Slave Firmware.....	9
Table 2: Terms, Abbreviations and Definitions.....	11
Table 3: References.....	11
Table 4: Names of Queues in CC-Link Slave Firmware.....	15
Table 5: Meaning of Source- and Destination-related Parameters. ....	15
Table 6: Meaning of Destination-Parameter ulDest.Parameters. ....	17
Table 7 Example for correct Use of Source- and Destination-related parameters.: ....	19
Table 8: Input Data Image .....	25
Table 9: Output Data Image.....	25
Table 10: General Structure of Packets for non-cyclic Data Exchange.....	27
Table 11: Status and Error Codes.....	30
Table 12: Channel Mailboxes.....	31
Table 13: Common Status Structure Definition .....	33
Table 14: Communication State of Change .....	34
Table 15: Meaning of Communication Change of State Flags.....	35
Table 16: ST1 – Data Transfer from Master to Slave Station .....	41
Table 17: Allowed Values of Master Station (User Application Program) .....	41
Table 18: Allowed Values of Master Station User Application Program Error Check.....	41
Table 19: Allowed Values of Refresh Startup .....	42
Table 20: Allowed Values of Transient Data Status .....	42
Table 21: Allowed Values of Transient Data Reception Enable.....	42
Table 22: Allowed Values of Protocol Version .....	42
Table 23: Allowed Values of Master Station Type.....	43
Table 24: ST1 – Data Transfer from Slave to Master Station .....	44
Table 25: Allowed Values of Fuse Status .....	44
Table 26: Allowed Values of Unit Error/Invalid No. of Points .....	45
Table 27: Allowed Values of Refresh Receive .....	45
Table 28: Allowed Values of Parameter Receive.....	45
Table 29: Allowed Values of Switch Change Detection .....	45
Table 30: Allowed Values of Cyclic Transmission Flag.....	46
Table 31: Allowed Values of Watchdog Timer Error .....	46
Table 32: ST2 – Data Transfer from Master to Slave Station .....	46
Table 33: RY Information Transmission Points .....	47
Table 34: RWw Information Transmission Points .....	47
Table 35: ST2 – Data Transfer from Slave to Master Station .....	48
Table 36: Allowed Values of Transient Data Status .....	48
Table 37: Allowed Values of Transient Receive.....	48
Table 38: Allowed Values of Transient Type.....	49
Table 39: Allowed Values of Transmission Status .....	49
Table 40: Allowed Values of Extended cycle setting.....	49
Table 41: Communication Control Block.....	51
Table 42: Overview about Essential Functionality.....	52
Table 43: Meaning and allowed Values for Warmstart-Parameters.....	54
Table 44: Available Baud Rate Values.....	55
Table 45: Input and Output Data for Remote I/O Device .....	56
Table 46: Input and Output Data for Remote Device Station with One Occupied Station.....	56
Table 47: Input and Output Data for Remote Device Station with Two Occupied Stations .....	56
Table 48: Input and Output Data for Remote Device Station with Three Occupied Stations .....	56
Table 49: Input and Output Data for Remote Device Station with Four Occupied Stations .....	57
Table 50: Input and Output Data for Remote Device Station with One Occupied Stations, Single Setting .....	58
Table 51: Input and Output Data for Remote Device Station with Two Occupied Stations, Single Setting .....	58
Table 52: Input and Output Data for Remote Device Station with Three Occupied Stations, Single Setting .....	58
Table 53: Input and Output Data for Remote Device Station with Four Occupied Stations, Single Setting .....	58
Table 54: Input and Output Data for Remote Device Station with One Occupied Station, Double Setting .....	59
Table 55: Input and Output Data for Remote Device Station with Two Occupied Stations, Double Setting.....	59
Table 56: Input and Output Data for Remote Device Station with Three Occupied Stations, Double Setting .....	59
Table 57: Input and Output Data for Remote Device Station with Four Occupied Stations, Double Setting .....	59
Table 58: Input and Output Data for Remote Device Station with One Occupied Station, Quadruple Setting .....	60
Table 59: Input and Output Data for Remote Device Station with Two Occupied Stations, Quadruple Setting .....	60
Table 60: Input and Output Data for Remote Device Station with Three Occupied Stations, Quadruple Setting... ..	60
Table 61: Input and Output Data for Remote Device Station with Four Occupied Stations, Quadruple Setting.....	60
Table 62: Input and Output Data for Remote Device Station with One Occupied Station, Octuple Setting .....	61
Table 63: Input and Output Data for Remote Device Station with Two Occupied Stations, Octuple Setting.....	61
Table 64: Input and Output Data for Remote Device Station with Three Occupied Stations, Octuple Setting .....	61

Table 65: Input and Output Data for Remote Device Station with Four Occupied Stations, Octuple Setting .....	61
Table 66: CC-Link APS-Task Process Queue .....	65
Table 67: Overview over the Packets of the APS-Task of the CC-Link Slave Protocol Stack.....	65
Table 68: CCLINK_APS_PCK_WARMSTART_REQ_T – Set Warmstart Parameter Request.....	70
Table 69: CCLINK_APS_PCK_WARMSTART_CNF_T – Set Warmstart Parameter Confirmation .....	74
Table 70: CCLINK_APS_PCK_WARMSTART_CNF – Packet Status/Error .....	75
Table 71: Possible Values for Parameter ulIoTypesPoints.....	77
Table 72: CCLINK_APS_PCK_SET_CONFIGURATION_REQ_T – Set Warmstart Parameter Request.....	80
Table 73: CCLINK_APS_PCK_SET_CONFIGURATION_CNF_T – Set Warmstart Parameter Confirmation.....	81
Table 74: CC-Link APS-Task Process Queue .....	82
Table 75: Overview over the Packets of the CC-Link Slave-Task k of the CC-Link Slave Protocol Stack .....	83
Table 76: CCLINK_SLAVE_PACKET_INITIALIZE_REQ_T – Initialization of CC-Link Slave Request .....	85
Table 77: CCLINK_SLAVE_PACKET_INITIALIZE_CNF_T – Initialization of CC-Link Slave Confirmation .....	86
Table 78: CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_T – Register Application Request .....	88
Table 79: CCLINK_SLAVE_PACKET_APP_REGISTER_REQ – Packet Status/Error .....	88
Table 80: CCLINK_SLAVE_PACKET_APP_REGISTER_CNF_T – Register Application Confirmation .....	89
Table 81: CCLINK_SLAVE_PACKET_APP_REGISTER_CNF – Packet Status/Error .....	90
Table 82: CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_REQ_T – Get Buffer Handle Request.....	92
Table 83: CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_CNF_T – Get Buffer Handle Confirmation.....	94
Table 84: CCLINK_SLAVE_PACKET_CFG_BUS_PARAM_T - Bus Parameter Configuration.....	97
Table 85: CCLINK_SLAVE_PACKET_CFG_ADD_PARAM_T - Additional Configuration.....	97
Table 86: CCLINK_SLAVE_PACKET_SET_BUSPARAM_REQ_DATA_T – Set Bus Parameter Request .....	99
Table 87: CCLINK_SLAVE_PACKET_SET_BUSPARAM_CNF_T –Set Bus Parameter Confirmation .....	100
Table 88: CCLINK_SLAVE_PACKET_STARTSTOP_REQ_T – Start/Stop Communication Request .....	102
Table 89: CCLINK_SLAVE_PACKET_STARTSTOP_CNF_T – Start/Stop Communication Confirmation .....	104
Table 90: CCLINK_SLAVE_PACKET_GET_CCL_STATUS_REQ_T – Get CC-Link Status Request .....	106
Table 91: CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_T – Get CC-Link Status Confirmation .....	108
Table 92: CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_REQ_T – Change CC-Link Slave Status Request .....	111
Table 93: CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_T – Change CC-Link Slave Status Confirmation .....	114
Table 94: CCLINK_SLAVE_PACKET_GET_BUS_PARAM_REQ_T – Get Bus Parameter Request .....	115
Table 95: CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_T – Get Bus Parameter Confirmation.....	117
Table 96: CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_T – Change of State Indication .....	119
Table 97: CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_T – Change of State Response.....	122
Table 98: CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_REQ_T – Set Watchdog Fail Request .....	123
Table 99: CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_T – Set Watchdog Fail Confirmation .....	124
Table 100: Status/Error Codes CC-Link APS-Task.....	126
Table 101: Status/Error Codes CC-Link Slave-Task.....	128

# 1 Introduction

## 1.1 Abstract

This manual describes the application interface of the CC-Link Slave- stack, with the aim to support and lead you during the integration process of the given stack into your own application.

Stack development is based on Hilscher's Task Layer Reference Programming Model. This model defines the general template used to create a task including a combination of appropriate functions belonging to the same type of protocol layer. Furthermore, it defines of how different tasks have to communicate with each other in order to exchange data between each communication layer. This Reference Model is used by all programmers at Hilscher and shall be used by the developer when writing an application task on top of the stack.

## 1.2 System Requirements

This software package has the following environmental system requirements:

- netX-Chip as CPU hardware platform
- Operating system for task scheduling required

## 1.3 Intended Audience

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

- Knowledge of the programming language C
- Knowledge of the use of the real time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model
- Knowledge of the CC-Link Specification BAP-05026-J

## 1.4 Specifications

This stack has been written to meet the requirements outlined in the CC-Link specification Ver.2.00 BAP-05026-J.

### 1.4.1 Protocol Task System

To manage the CC-Link Slave implementation 2 tasks are involved into the system. To send packets to a task, the task main queue has to be identified. For the identifier for the tasks and there Queues are the following naming conversion:

Task Name	Queue Name	Description
CCLINK_SLAVE	"QUE_CCLSLAVE"	CC-Link Slave protocol task
CCLINK_APS	"QUE_CCLAPS"	CC-Link Slave application task

Table 1: Names of Tasks in CC-Link Slave Firmware

### 1.4.2 Technical Data

The data below applies to the CC-Link firmware and stack version V2.4.x. The firmware/stack support CC-Link Version 2.0 and 1.11

This firmware/stack has been written to meet the requirements outlined in the CC-Link specification Ver.2.00 BAP-05026-J.

#### Technical Data

Data for firmware/stack working according to CC-Link Version 2.0

Station Types	Remote device station, up to four occupied stations
Maximum input data	368 bytes
Maximum output data	368 bytes
Input data remote device station	112 bytes (RY) and 256 bytes (RWw)
Output data remote device station	112 bytes (RX) and 256 bytes (RWr)
Extension cycles	1, 2, 4, 8
Baud rates	156 KBit/s, 625 KBit/s, 2500 kBit/s, 5 MBit/s, 10 MBit/s

---

Data for firmware/stack working according to CC-Link Version 1.11

Station Types	Remote I/O station, Remote device station (Up to four occupied stations)
Maximum input data	48 bytes
Maximum output data	48 bytes
Input data remote I/O station	4 bytes (RY)
Output data remote I/O station	4 bytes (RX)
Input data remote device station	4 bytes (RY) and 8 bytes (RWw) per occupied station
Output data remote device station	4 bytes (RX) and 8 bytes (RWr) per occupied station
Baud rates	156 KBit/s, 625 KBit/s, 2500 kBit/s, 5 MBit/s, 10 MBit/s

### PCI

DMA Support for PCI targets	yes
-----------------------------	-----

### Firmware/stack available for netX

netX 50	yes
netX 100, netX 500	yes

### Configuration

Configuration by packet to transfer warmstart parameters

### Diagnostic

Firmware supports common and extended diagnostic in the dual-port-memory for loadable firmware

### Limitations

Intelligent Device Station not supported yet

## 1.5 Terms, Abbreviations and Definitions

Term	Description
AP	Application on top of the Stack
RX	Receive bit data (Remote I/O Station and Remote Device Station)
RY	Send bit data (Remote I/O Station and Remote Device Station)
RWr	Receive register data (Remote Device Station only)
RWw	Send register data (Remote Device Station only)

Table 2: Terms, Abbreviations and Definitions

All variables, parameters, and data used in this manual have the LSB/MSB ("Intel") data representation. This corresponds to the convention of the Microsoft C Compiler.

## 1.6 References

This document is based on the following specifications:

1	netX DPM Interface Manual, Hilscher GmbH
2	CC-Link Specification BAP-05026-J

Table 3: References

## **1.7 Legal Notes**

### **1.7.1 Copyright**

© 2006-2010 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

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

### **1.7.2 Important Notes**

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

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

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

### 1.7.3 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.

### 1.7.4 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 Fundamentals

### 2.1 General Access Mechanisms on netX Systems

This chapter explains the possible ways to access a Protocol Stack running on a netX system :

1. By accessing the Dual Port Memory Interface directly or via a driver.
2. By accessing the Dual Port Memory Interface via a shared memory.
3. By interfacing with the Stack Task of the Protocol Stack.

The picture below visualizes these three ways:

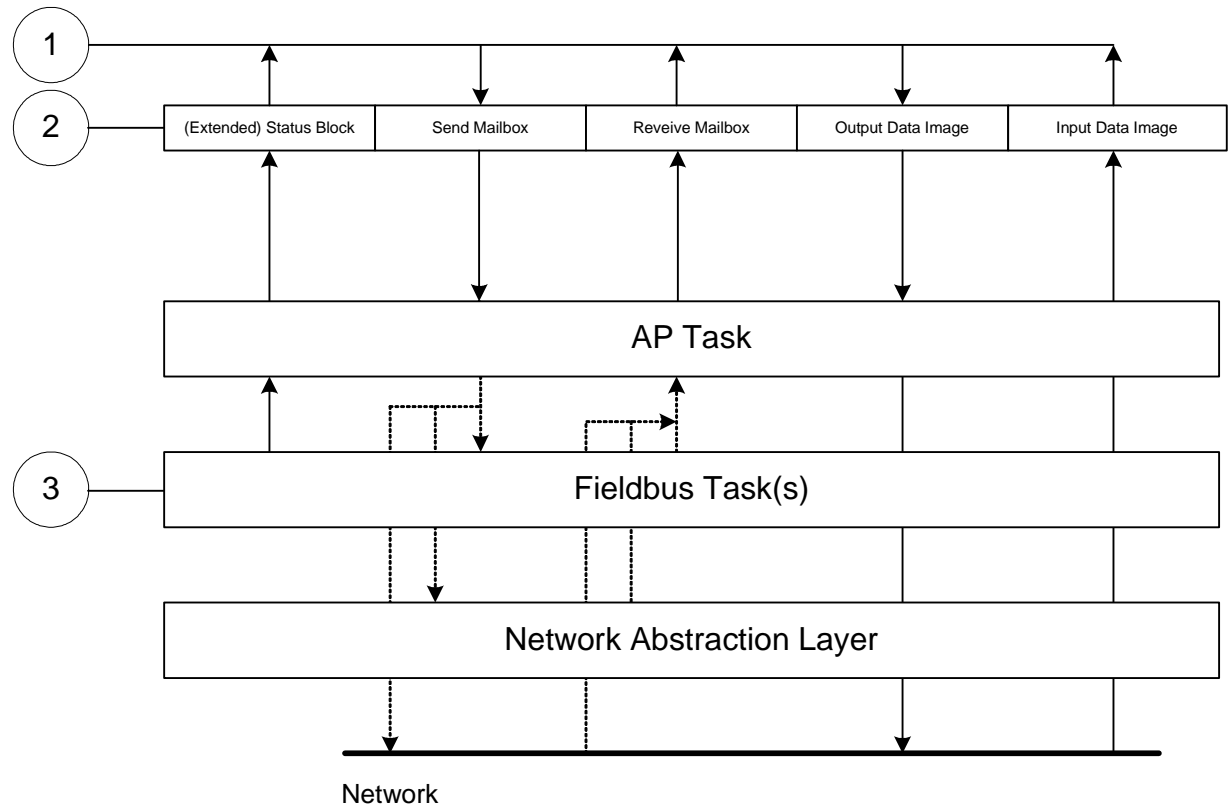


Figure 1 - The three different Ways to access a Protocol Stack running on a netX System

This chapter explains how to program the stack (alternative 3) correctly while the next chapter describes accessing the protocol stack via the dual-port memory interface according to alternative 1 (and 2, if the user application is executed on the netX chip in the context of the rcX operating system and uses the shared DPM). Finally, chapter 1 titled "" describes the entire interface to the protocol stack in detail.

Depending on you choose the stack-oriented approach or the Dual Port Memory-based approach, you will need either the information given in this chapter or those of the next chapter to be able to work with the set of functions described in chapter 5. All of those functions use the four parameters `ulDest`, `ulSrc`, `ulDestId` and `ulSrcId`. This chapter and the next one inform about how to work with these important parameters.

## 2.2 Accessing the Protocol Stack by Programming the AP Task's Queue

In general, programming the AP task or the stack has to be performed according to the rules explained in the Hilscher Task Layer Reference Manual. There you can also find more information about the variables discussed in the following.

### 2.2.1 Getting the Receiver Task Handle of the Process Queue

To get the handle of the process queue of the CC-Link Slave Protocol-Task or the CC-Link Slave Application-Task the Macro `TLR_QUE_IDENTIFY()` needs to be used. It is described in detail within section 10.1.9.3 of the Hilscher Task Layer Reference Model Manual. This macro delivers a pointer to the handle of the intended queue to be accessed (which is returned within the third parameter, `phQue`), if you provide it with the name of the queue (and an instance of your own task). The correct ASCII-queue names for accessing the CC-Link Slave Protocol-Task or the CC-Link Slave Application-Task, which you have to use as current value for the first parameter (`pszIdn`), is

ASCII Queue name	Description
"QUE_CCLSLAVE"	Name of the CC-Link Slave Protocol-Task process queue
"QUE_CCLAPS"	Name of the CC-Link Slave Application-Task process queue

Table 4: Names of Queues in CC-Link Slave Firmware

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the CC-Link Slave-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDBUFFER_FIFO/LIFO()` for sending a packet to the respective task.

### 2.2.2 Meaning of Source- and Destination-related Parameters

The meaning of the source- and destination-related parameters is explained in the following table:

Variable	Meaning
<code>ulDest</code>	Application mailbox used for confirmation
<code>ulSrc</code>	Queue handle returned by <code>TLR_QUE_IDENTIFY()</code> as described above.
<code>ulSrcId</code>	Used for addressing at a lower level

Table 5: Meaning of Source- and Destination-related Parameters.

For more information about programming the AP task's stack queue, please refer to the Hilscher Task Layer Reference Model Manual. Especially the following sections might be of interest in this context:

1. Chapter 7 "Queue-Packets"
2. Section 10.1.9 "Queuing Mechanism"

## 2.3 Accessing the Protocol Stack via the Dual Port Memory Interface

This chapter defines the application interface of the CC-Link Slave Stack.

### 2.3.1 Communication via Mailboxes

The mailbox of each communication channel has two areas that are used for non-cyclic message transfer to and from the netX.

- **Send Mailbox**  
Packet transfer from host system to netX firmware
- **Receive Mailbox**  
Packet transfer from netX firmware to host system

For more details about acyclic data transfer via mailboxes, see section 3.2. [Acyclic Data \(Mailboxes\)](#) in this context, is described in detail in section 3.2.1 "[General Structure of Messages or Packets for Non-Cyclic Data Exchange](#)" while the possible codes that may appear are listed in section 3.2.2. "[Status & Error Codes](#)".

However, this section concentrates on correct addressing the mailboxes.

### 2.3.2 Using Source and Destination Variables correctly

#### 2.3.2.1 How to use `ulDest` for Addressing `rcX` and the `netX` Protocol Stack by the System and Channel Mailbox

The preferred way to address the `netX` operating system `rcX` is through the system mailbox; the preferred way to address a protocol stack is through its channel mailbox. All mailboxes, however, have a mechanism to route packets to a communication channel or the system channel, respectively. Therefore, the destination identifier `ulDest` in a packet header has to be filled in according to the targeted receiver. See the following example:

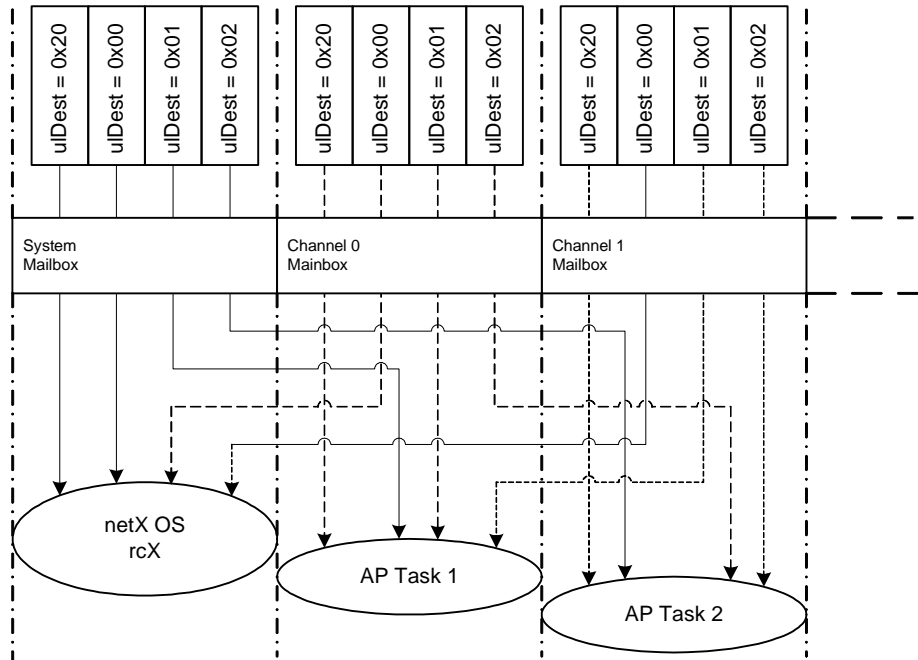


Figure 2 - Use of `ulDest` in Channel and System Mailbox

For use in the destination queue handle, the tasks have been assigned to hexadecimal numerical values as described in the following table:

<code>ulDest</code>	Description
<code>0x</code>	Packet is passed to the <code>netX</code> operating system <code>rcX</code>
<code>0x0001</code>	Packet is passed to communication channel 0
<code>0x0002</code>	Packet is passed to communication channel 1
<code>0x0003</code>	Packet is passed to communication channel 2
<code>0x0004</code>	Packet is passed to communication channel 3
<code>0x0020</code>	Packet is passed to communication channel of the mailbox
else	Reserved, do not use

Table 6: Meaning of Destination-Parameter `ulDest`.Parameters.

The figure and the table above both show the use of the destination identifier `ulDest`.

A remark on the special channel identifier `0x0020` (= *Channel Token*). The Channel Token is valid for any mailbox. That way the application uses the same identifier for all packets without actually knowing which mailbox or communication channel is applied. The packet stays 'local'. The system mailbox is a little bit different, because it is used to communicate to the netX operating system rcX. The rcX has its own range of valid commands codes and differs from a communication channel.

Unless there is a reply packet, the netX operating system returns it to the same mailbox the request packet went through. Consequently, the host application has to return its reply packet to the mailbox the request was received from.

### 2.3.2.2 How to use `ulSrc` and `ulSrcId`

Generally, a netX protocol stack can be addressed through its communication channel mailbox. The example below shows how a host application addresses a protocol stack running in the context of a netX chip. The application is identified by a number (#444 in this example). The application consists of three processes identified by the numbers #11, #22 and #33. These processes communicate through the channel mailbox with the AP task of the protocol stack. Have a look at the following figure:

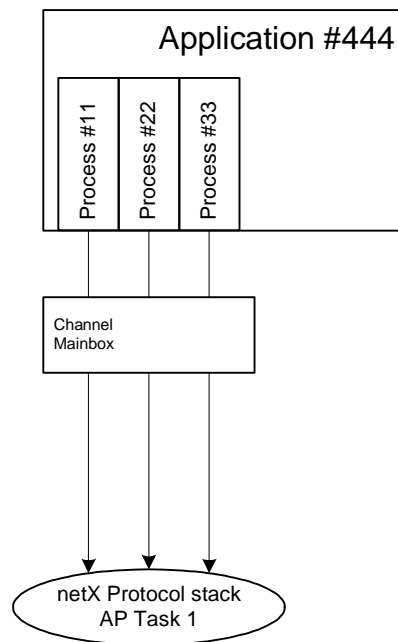


Figure 3 - Using `ulSrc` and `ulSrcId`

**Example:**

This example applies to command messages initiated by a process in the context of the host application. If the process #22 sends a packet through the channel mailbox to the AP task, the packet header has to be filled in as follows:

Object	Variable Name	Numeric Value	Explanation
Destination Queue Handle	ulDest	= 32 (0x0020)	This value needs always to be set to 0x0020 (the channel token) when accessing the protocol stack via the local communication channel mailbox.
Source Queue Handle	ulSrc	= 444	Denotes the host application (#444).
Destination Identifier	ulDestId	= 0	In this example, it is not necessary to use the destination identifier.
Source Identifier	ulSrcId	= 22	Denotes the process number of the process within the host application and needs therefore to be supplied by the programmer of the host application.

*Table 7 Example for correct Use of Source- and Destination-related parameters.:*

For packets through the channel mailbox, the application uses 32 (= 0x20, *Channel Token*) for the destination queue handler *ulDest*. The source queue handler *ulSrc* and the source identifier *ulSrcId* are used to identify the originator of a packet. The destination identifier *ulDestId* can be used to address certain resources in the protocol stack. It is not used in this example. The source queue handler *ulSrc* has to be filled in. Therefore, its use is mandatory; the use of *ulSrcId* is optional.

The netX operating system passes the request packet to the protocol stack's AP task. The protocol stack then builds a reply to the packet and returns it to the mailbox. The application has to make sure that the packet finds its way back to the originator (process #22 in the example).

### 2.3.2.3 How to Route rcX Packets

To route an rcX packet the source identifier *ulSrcId* and the source queues handler *ulSrc* in the packet header hold the identification of the originating process. The router saves the original handle from *ulSrcId* and *ulSrc*. The router uses a handle of its own choices for *ulSrcId* and *ulSrc* before it sends the packet to the receiving process. That way the router can identify the corresponding reply packet and matches the handle from that packet with the one stored earlier. Now the router replaces its handles with the original handles and returns the packet to the originating process.

### 2.3.3 Obtaining useful Information about the Communication Channel

A communication channel represents a part of the Dual Port Memory and 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 describes a procedure how to obtain useful information for accessing the communication channel(s) of your netX device and to check if it is ready for correct operation.

Proceed as follows:

- 1) Start with reading the channel information block within the system channel (usually starting at address 0x0030).
- 2) Then you should check the hardware assembly options of your netX device. They are located within the system information block following offset 0x0010 and stored as data type `UINT16`. The following table explains the relationship between the offsets and the corresponding xC Ports of the netX device:

0x0010	Hardware Assembly Options for xC Port[0]
0x0012	Hardware Assembly Options for xC Port[1]
0x0014	Hardware Assembly Options for xC Port[2]
0x0016	Hardware Assembly Options for xC Port[3]

Check each of the hardware assembly options whether its value has been set to `RCX_HW_ASSEMBLY_CCLINK = 0x0070`. If true, this denotes that this xCPort is suitable for running the CC-Link Slave protocol stack. Otherwise, this port is designed for another communication protocol. In most cases, xC Port[2] will be used for field bus systems, while xC Port[0] and xC Port[1] are normally used for Ethernet communication.

- 3) You can find information about the corresponding communication channel (0...3) under the following addresses:

0x0050	Communication Channel 0
0x0060	Communication Channel 1
0x0070	Communication Channel 2
0x0080	Communication Channel 3

In devices which support only one communication system which is usually the case (either a single field bus system or a single standard for Industrial-Ethernet communication), always communication channel 0 will be used. In devices supporting more than one communication system you should also check the other communication channels.

- 4) There you can find such information as the ID (containing channel number and port number) of the communication channel, the size and the location of the handshake cells, the overall number of blocks within the communication channel and the size of the channel in bytes. Evaluate this information precisely in order to access the communication channel correctly.

The information is delivered as follows:

Size of Channel in Bytes

Address	Data Type	Description
0x0050	UINT8	Channel Type = COMMUNICATION (must have the fixed value <code>define RCX_CHANNEL_TYPE_COMMUNICATION = 0x05</code> )
0x0051	UINT8	ID (Channel Number, Port Number)
0x0052	UINT8	Size / Position Of Handshake Cells
0x0053	UINT8	Total Number Of Blocks Of This Channel
0x0054	UINT32	Size Of Channel In Bytes
0x0058	UINT8[8]	Reserved (set to zero)

These addresses correspond to communication channel 0, for communication channels 1, 2 and 3 you have to add an offset of 0x0010, 0x0020 or 0x0030 to the address values, respectively.

## 2.4 Client/Server Mechanism

### 2.4.1 Application as Client

The host application may send request packets to the netX firmware at any time (transition 1 ⇒ 2). Depending on the protocol stack running on the netX, parallel packets are not permitted (see protocol specific manual for details). The netX firmware sends a confirmation packet in return, signaling success or failure (transition 3 ⇒ 4) while processing the request.

The host application has to register with the netX firmware in order to receive indication packets (transition 5 ⇒ 6). Depending on the protocol stack, this is done either implicit (if application opens a TCP/UDP socket) or explicit (if application wants to receive unsolicited DPV1 packets). Details on when and how to register for certain events is described in the protocol specific manual. Depending on the command code of the indication packet, a response packet to the netX firmware may or may not be required (transition 7 ⇒ 8).

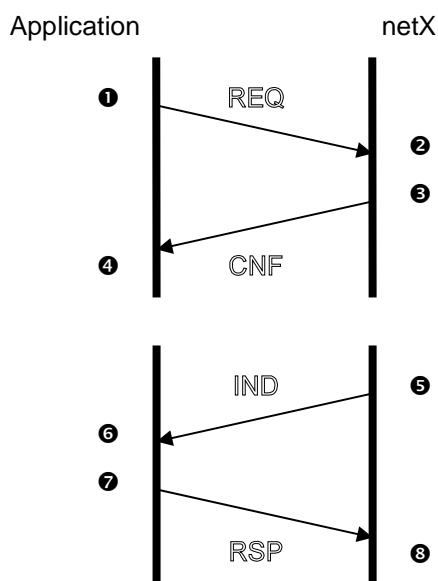


Figure 4: Transition Chart Application as Client

- ① ② The host application sends request packets to the netX firmware.
- ③ ④ The netX firmware sends a confirmation packet in return.
- ⑤ ⑥ The host application receives indication packets from the netX firmware.
- ⑦ ⑧ The host application sends response packet to the netX firmware (may not be required).

REQ Request                      CNF Confirmation

IND Indication                    RSP Response

### 2.4.2 Application as Server

The host application has to register with the netX firmware in order to receive indication packets. Depending on the protocol stack, this is done either implicit (if application opens a TCP/UDP socket) or explicit (if application wants to receive unsolicited DPV1 packets). Details on when and how to register for certain events is described in the protocol specific manual.

When an appropriate event occurs and the host application is registered to receive such a notification, the netX firmware passes an indication packet through the mailbox (transition 1 ⇒ 2). The host application is expected to send a response packet back to the netX firmware (transition 3 ⇒ 4).

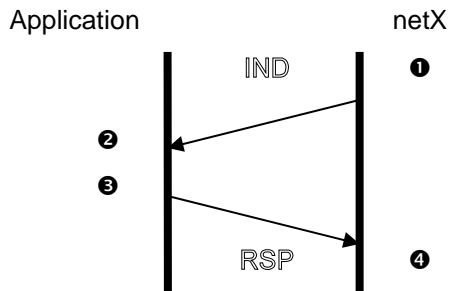


Figure 5: Transition Chart Application as Server

- ❶ ❷ The netX firmware passes an indication packet through the mailbox.
- ❸ ❹ The host application sends response packet to the netX firmware.

IND Indication                      RSP Response

## 3 Dual-Port Memory

All data in the dual-port memory is structured in blocks. According to their functions, these blocks use different data transfer mechanisms. For example, data transfer through mailboxes uses a synchronized handshake mechanism between host system and netX firmware. The same is true for IO data images, when a buffered handshake mode is configured. Other blocks, like the status block, are read by the host application and use no synchronization mechanism.

Types of blocks in the dual-port memory are outlined below:

- **Mailbox**  
transfer non-cyclic messages or packages with a header for routing information
- **Data Area**  
holds the process image for cyclic I/O data or user defined data structures
- **Control Block**  
is used to signal application related state to the netX firmware
- **Status Block**  
holds information regarding the current network state
- **Change of State**  
collection of flags that initiate execution of certain commands or signal a change of state

### 3.1 Cyclic Data (Input/Output Data)

The input block holds the process data image received **from** the network whereas the output block holds data sent **to** the network

Process data transfer through the data blocks can be synchronized by using a handshake mechanism (configurable). If in uncontrolled mode, the protocol stack updates the process data in the input and output data image in the dual-port memory for each valid bus cycle. No handshake bits are evaluated and no buffers are used. The application can read or write process data at any given time without obeying the synchronization mechanism otherwise carried out via handshake location. This transfer mechanism is the simplest method of transferring process data between the protocol stack and the application. This mode can only guarantee data consistency over a byte.

For the controlled / buffered mode, the protocol stack updates the process data in the internal input buffer for each valid bus cycle. Each IO block uses handshake bits for access synchronization. Input and output data block handshake operates independently from each other. When the application toggles the input handshake bit, the protocol stack copies the data from the internal buffer into the input data image of the dual-port memory. Now the application can copy data from the dual-port memory and then give control back to the protocol stack by toggling the appropriate input handshake bit. When the application/driver toggles the output handshake bit, the protocol stack copies the data from the output data image of the dual-port memory into the internal buffer. From there the data is transferred to the network. The protocol stack toggles the handshake bits back, indicating to the application that the transfer is finished and a new data exchange cycle may start. This mode guarantees data consistency over both input and output area.

### 3.1.1 Input Process Data

The input data block is used by field bus and industrial Ethernet protocols that utilize a cyclic data exchange mechanism. The input data image is used to receive cyclic data **from** the network.

The default size of the input data image is 5760 byte. However, not all available space is actually used by the protocol stack. Depending on the specific protocol, the area actually available for user data might be much smaller than 5760 byte. An input data block may or may not be available in the dual-port memory. It is always available in the default memory map (see the *netX Dual-Port Memory Manual*).

Input Data Image			
Offset	Type	Name	Description
0x2680	UINT8	abPd0Input [ 5760 ]	Input Data Image Cyclic Data From The Network

Table 8: Input Data Image

### 3.1.2 Output Process Data

The output data block is used by field bus and industrial Ethernet protocols that utilize a cyclic data exchange mechanism. The output data Image is used to send cyclic data from the host **to** the network.

The default size of the output data image is 5760 byte. However, not all available space is actually used by the protocol stack. Depending on the specific protocol, the area actually available for user data might be much smaller than 5760 byte. An output data block may or may not be available in the dual-port memory. It is always available in the default memory map (see *netX DPM Manual*).

Output Data Image			
Offset	Type	Name	Description
0x1000	UINT8	abPd0Output [ 5760 ]	Output Data Image Cyclic Data To The Network

Table 9: Output Data Image

## 3.2 Acyclic Data (Mailboxes)

The mailbox of each communication channel has two areas that are used for non-cyclic message transfer to and from the netX processor.

- **Send Mailbox**  
Packet transfer from host system to firmware
- **Receive Mailbox**  
Packet transfer from firmware to host system

The send and receive mailbox areas are used by field bus and industrial Ethernet protocols providing a non-cyclic data exchange mechanism. Another use of the mailbox system is to allow access to the firmware running on the netX chip itself for diagnostic and identification purposes.

The send mailbox is used to transfer acyclic data **to** the network or **to** the firmware. The receive mailbox is used to transfer acyclic data **from** the network or **from** the firmware.

A send/receive mailbox may or may not be available in the communication channel. It depends on the function of the firmware whether or not a mailbox is needed. The location of the system mailbox and the channel mailbox is described in the *netX DPM Interface Manual*.

---

**Note:** Each mailbox can hold one packet at a time. The netX firmware stores packets that are not retrieved by the host application in a packet queue. This queue has limited space and may fill up so new packets maybe lost. To avoid these data loss situations, it is strongly recommended to empty the mailbox frequently, even if packets are not expected by the host application. Unexpected command packets should be returned to the sender with an *Unknown Command* in the status field; unexpected reply messages can be discarded.

---

### 3.2.1 General Structure of Messages or Packets for Non-Cyclic Data Exchange

The non-cyclic packets through the netX mailbox have the following structure:

Structure Information				
Area	Variable	Type	Value / Range	Description
Head	Structure Information			
	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)
	ulId	UINT32		Packet Identification As Unique Number
	ulSta	UINT32		Status / Error Code
	ulCmd	UINT32		Command / Response
	ulExt	UINT32		Extension Flags
	ulRout	UINT32		Routing Information
Data	Structure Information			
	...	...		User Data Specific To The Command

Table 10: General Structure of Packets for non-cyclic Data Exchange.

Some of the fields are mandatory; some are conditional; others are optional. However, the size of a packet is always at least 10 double-words or 40 bytes. Depending on the command, a packet may or may not have a data field. If present, the content of the data field is specific to the command, respectively the reply.

#### Destination Queue Handle

The *ulDest* field identifies a task queue in the context of the netX firmware. The task queue represents the final receiver of the packet and is assigned to a protocol stack. The *ulDest* field has to be filled out in any case. Otherwise, the netX operating system cannot route the packet. This field is mandatory.

**Source Queue Handle**

The *uSrc* field identifies the sender of the packet. In the context of the netX firmware (inter-task communication) this field holds the identifier of the sending task. Usually, a driver uses this field for its own handle, but it can hold any handle of the sending process. Using this field is mandatory. The receiving task does not evaluate this field and passes it back unchanged to the originator of the packet.

**Destination Identifier**

The *uDestId* field identifies the destination of an unsolicited packet from the netX firmware to the host system. It can hold any handle that helps to identify the receiver. Therefore, its use is mandatory for unsolicited packets. The receiver of unsolicited packets has to register for this.

**Source Identifier**

The *uSrcId* field identifies the originator of a packet. This field is used by a host application, which passes a packet from an external process to an internal netX task. The *uSrcId* field holds the handle of the external process. When netX operating system returns the packet, the application can identify the packet and returns it to the originating process. The receiving task on the netX does not evaluate this field and passes it back unchanged. For inter-task communication, this field is not used.

**Length of Data Field**

The *uLen* field holds the size of the data field in bytes. It defines the total size of the packet's payload that follows the packet's header. The size of the header is not included in *uLen*. So the total size of a packet is the size from *uLen* plus the size of packet's header. Depending on the command, a data field may or may not be present in a packet. If no data field is included, the length field is set to zero.

**Identifier**

The *uId* field is used to identify a specific packet among others of the same kind. That way the application or driver can match a specific reply or confirmation packet to a previous request packet. The receiving task does not change this field and passes it back to the originator of the packet. Its use is optional in most of the cases. However, it is mandatory for sequenced packets.

Example: Downloading big amounts of data that does not fit into a single packet. For a sequence of packets the identifier field is incremented by one for every new packet.

**Status / Error Code**

The *ulSta* field is used in response or confirmation packets. It informs the originator of the packet about success or failure of the execution of the command. The field may be also used to hold status information in a request packet.

**Command / Response**

The *ulCmd* field holds the command code or the response code, respectively. The command/response is specific to the receiving task. If a task is not able to execute certain commands, it will return the packet with an error indication. A command is always even (the least significant bit is zero). In the response packet, the command code is incremented by one indicating a confirmation to the request packet.

**Extension Flags**

The extension field *ulExt* is used for controlling packets that are sent in a sequenced manner. The extension field indicates the first, last or a packet of a sequence. If sequencing is not required, the extension field is not used and set to zero.

**Routing Information**

The *ulRout* field is used internally by the netX firmware only. It has no meaning to a driver type application and therefore set to zero.

**User Data Field**

This field contains data related to the command specified in *ulCmd* field. Depending on the command, a packet may or may not have a data field. The length of the data field is given in the *ulLen* field.

### 3.2.2 Status & Error Codes

The following status and error codes can be returned in `ulSta`:

Status and Error Codes		
Code (Symbolic Constant)	Numerical Value	Meaning
RCX_S_OK	0x	SUCCESS, STATUS OKAY
RCX_S_QUE_UNKNOWN	0xC02B0001	UNKNOWN QUEUE
RCX_S_QUE_INDEX_UNKNOWN	0xC02B0002	UNKNOWN QUEUE INDEX
RCX_S_TASK_UNKNOWN	0xC02B0003	UNKNOWN TASK
RCX_S_TASK_INDEX_UNKNOWN	0xC02B0004	UNKNOWN TASK INDEX
RCX_S_TASK_HANDLE_INVALID	0xC02B0005	INVALID TASK HANDLE
RCX_S_TASK_INFO_IDX_UNKNOWN	0xC02B0006	UNKNOWN INDEX
RCX_S_FILE_XFR_TYPE_INVALID	0xC02B0007	INVALID TRANSFER TYPE
RCX_S_FILE_REQUEST_INCORRECT	0xC02B0008	INVALID FILE REQUEST
RCX_S_UNKNOWN_DESTINATION	0xC0000005	UNKNOWN DESTINATION
RCX_S_UNKNOWN_DESTINATION_ID	0xC0000006	UNKNOWN DESTINATION ID
RCX_S_INVALID_LENGTH	0xC0000007	INVALID LENGTH
RCX_S_UNKNOWN_COMMAND	0xC0000004	UNKNOWN COMMAND
RCX_S_INVALID_EXTENSION	0xC0000008	INVALID EXTENSION

Table 11: Status and Error Codes.

### 3.2.3 Differences between System and Channel Mailboxes

The mailbox system on netX provides a non-cyclic data transfer channel for field bus and industrial Ethernet protocols. Another use of the mailbox is allowing access to the firmware running on the netX chip itself for diagnostic purposes. There is always a send and a receive mailbox. Send and receive mailboxes utilize handshake bits to synchronize these data or diagnostic packages through the mailbox. There is a pair of handshake bits for both the send and receive mailbox.

The netX operating system rcX only uses the system mailbox.

- The *system mailbox*, however, has a mechanism to route packets to a communication channel.
- A *channel mailbox* passes packets to its own protocol stack only.

### 3.2.4 Send Mailbox

The send mailbox area is used by protocols utilizing a non-cyclic data exchange mechanism. Another use of the mailbox system is to provide access to the firmware running on the netX chip itself. The **send** mailbox is used to transfer non-cyclic data **to** the network or **to** the protocol stack.

The size is 1596 bytes for the send mailbox in the default memory layout. The mailbox is accompanied by counters that hold the number of packages that can be accepted.

### 3.2.5 Receive Mailbox

The receive mailbox area is used by protocols utilizing a non-cyclic data exchange mechanism. Another use of the mailbox system is to provide access to the firmware running on the netX chip itself. The **receive** mailbox is used to transfer non-cyclic data **from** the network or **from** the protocol stack.

The size is 1596 bytes for the receive mailbox in the default memory layout. The mailbox is accompanied by counters that hold the number of waiting packages (for the receive mailbox).

### 3.2.6 Channel Mailboxes (Details of Send and Receive Mailboxes)

Master Status			
Offset	Type	Name	Description
0x0200	UINT16	usPackagesAccepted	Packages Accepted Number of Packages that can be Accepted
0x0202	UINT16	usReserved	Reserved Set to 0
0x0204	UINT8	abSendMbx[ 1596 ]	Send Mailbox Non Cyclic Data To The Network or to the Protocol Stack
0x0840	UINT16	usWaitingPackages	Packages waiting Counter of packages that are waiting to be processed
0x0842	UINT16	usReserved	Reserved Set to 0
0x0844	UINT8	abRecvMbx[ 1596 ]	Receive Mailbox Non Cyclic Data <b>from</b> the network or <b>from</b> the protocol stack

Table 12: Channel Mailboxes.

## Channel Mailboxes Structure

```
typedef struct tagNETX_SEND_MAILBOX_BLOCK
{
  UINT16 usPackagesAccepted;
  UINT16 usReserved;
  UINT8 abSendMbx[ 1596 ];
} NETX_SEND_MAILBOX_BLOCK;
typedef struct tagNETX_RECV_MAILBOX_BLOCK
{
  UINT16 usWaitingPackages;
  UINT16 usReserved;
  UINT8 abRecvMbx[ 1596 ];
} NETX_RECV_MAILBOX_BLOCK;
```

## 3.3 Status

A status block is present within the communication channel. It contains information about network and task related issues. In some respects, status and control block are used together in order to exchange information between host application and netX firmware. The application reads a status block whereas the control block is written by the application. Both status and control block have registers that use the *Change of State* mechanism (see also section 2.2.1 of the *netX Dual-Port-Memory manual*).

### 3.3.1 Common Status

The Common Status Block contains information that is the same for all communication channels. The start offset of this block depends on the size and location of the preceding blocks. The status block is always present in the dual-port memory.

#### 3.3.1.1 All Implementations

The structure outlined below is common to all protocol stacks.

#### Common Status Structure Definition

Common Status			
Offset	Type	Name	Description
0x0010	UINT32	ulCommunicationCOS	<u>Communication Change of State</u> READY, RUN, RESET REQUIRED, NEW, CONFIG AVAILABLE, CONFIG LOCKED
0x0014	UINT32	ulCommunicationState	<u>Communication State</u> NOT CONFIGURED, STOP, IDLE, OPERATE
0x0018	UINT32	ulCommunicationError	<u>Communication Error</u> Unique Error Number According to Protocol Stack
0x001C	UINT16	usVersion	<u>Version</u> Version Number of this Diagnosis Structure
0x001E	UINT16	usWatchdogTime	<u>Watchdog Timeout</u>

			Configured Watchdog Time
<b>0x0020</b>	UINT16	usHandshakeMode	Handshake Mode Process Data Transfer Mode (see netX DPM Interface Manual)
<b>0x0022</b>	UINT16	usReserved	Reserved Set to 0
<b>0x0024</b>	UINT32	ulHostWatchdog	<u>Host Watchdog</u> Joint Supervision Mechanism Protocol Stack Writes, Host System Reads
<b>0x0028</b>	UINT32	ulErrorCount	<u>Error Count</u> Total Number of Detected Error Since Power-Up or Reset
<b>0x002C</b>	UINT32	ulErrorLogInd	<u>Error Log Indicator</u> Total Number Of Entries In The Error Log Structure (not supported yet)
<b>0x0030</b>	UINT32	ulReserved[2]	<u>Reserved</u> Set to 0

Table 13: Common Status Structure Definition

### Common Status Block Structure Reference

```
typedef struct NETX_COMMON_STATUS_BLOCK_Ttag
{
    UINT32    ulCommunicationCOS;
    UINT32    ulCommunicationState;
    UINT32    ulCommunicationError;
    UINT16    usVersion;
    UINT16    usWatchdogTime;
    UINT16    ausReserved[2];
    UINT32    ulHostWatchdog;
    UINT32    ulErrorCount;
    UINT32    ulErrorLogInd;
    UINT32    ulReserved[2];
    union
    {
        {
            NETX_MASTER_STATUS_T    tMasterStatus;    /* for master implementation */
            UINT32                    aulReserved[6];    /* otherwise reserved */
        } unStackDepended;
    }
} NETX_COMMON_STATUS_BLOCK_T;
```

## Common Status Block Structure Reference

```
typedef struct NETX_COMMON_STATUS_BLOCK_Ttag
{
  UUINT32    ulCommunicationCOS;
  UUINT32    ulCommunicationState;
  UUINT32    ulCommunicationError;
  UUINT16    usVersion;
  UUINT16    usWatchdogTime;
  UUINT16    ausReserved[2];
  UUINT32    ulHostWatchdog;
  UUINT32    ulErrorCount;
  UUINT32    ulErrorLogInd;
  UUINT32    ulReserved[2];
  union
  {
    {
      NETX_MASTER_STATUS_T    tMasterStatus;    /* for master implementation */
      UUINT32                  aulReserved[6];  /* otherwise reserved      */
    } unStackDepended;
  }
} NETX_COMMON_STATUS_BLOCK_T;
```

### Communication Change of State (All Implementations)

The communication change of state register contains information about the current operating status of the communication channel and its firmware. Every time the status changes, the netX protocol stack toggles the *netX Change of State Command* flag in the netX communication flags register (see section 3.2.2.1 of the netX DPM Interface Manual). The application then has to toggle the *netX Change of State Acknowledge* flag back acknowledging the new state (see section 3.2.2.2 of the netX DPM Interface Manual).

ulCommunicationCOS - netX writes, Host reads		
Bit	Short name	Name
D31..D7	unused, set to zero	
D6	Restart Required Enable	RCX_COMM_COS_RESTART_REQUIRED_ENABLE
D5	Restart Required	RCX_COMM_COS_RESTART_REQUIRED
D4	Configuration New	RCX_COMM_COS_CONFIG_NEW
D3	Configuration Locked	RCX_COMM_COS_CONFIG_LOCKED
D2	Bus On	RCX_COMM_COS_BUS_ON
D1	Running	RCX_COMM_COS_RUN
D0	Ready	RCX_COMM_COS_READY

Table 14: Communication State of Change

**Communication Change of State Flags (netX System ⇒ Application)**

Bit	Definition / Description
0	Ready (RCX_COMM_COS_READY) 0 - ... 1 - The <i>Ready</i> flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the <i>Running</i> flag is set, too.
1	Running (RCX_COMM_COS_RUN) 0 - ... 1 -The <i>Running</i> flag is set when the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the <i>Ready</i> flag and the <i>Running</i> flag are set.
2	Bus On (RCX_COMM_COS_BUS_ON) 0 - ... 1 -The <i>Bus On</i> flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network; if cleared, the permission was denied and the protocol stack will not open network connections.
3	Configuration Locked (RCX_COMM_COS_CONFIG_LOCKED) 0 - ... 1 -The <i>Configuration Locked</i> flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the <i>Lock Configuration</i> flag in the control block (see page 51).
4	Configuration New (RCX_COMM_COS_CONFIG_NEW) 0 - ... 1 -The <i>Configuration New</i> flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the <i>Restart Required</i> flag.
5	Restart Required (RCX_COMM_COS_RESTART_REQUIRED) 0 - ... 1 -The <i>Restart Required</i> flag is set when the channel firmware requests to be restarted. This flag is used together with the <i>Restart Required Enable</i> flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
6	Restart Required Enable (RCX_COMM_COS_RESTART_REQUIRED_ENABLE) 0 - ... 1 - The <i>Restart Required Enable</i> flag is used together with the <i>Restart Required</i> flag above. If set, this flag enables the execution of the <i>Restart Required</i> command in the netX firmware (for details on the <i>Enable</i> mechanism see section 2.3.2 of the netX DPM Interface Manual)).
7 ... 31	Reserved, set to 0

Table 15: Meaning of Communication Change of State Flags

**Communication State (All Implementations)**

The communication state field contains information regarding the current network status of the communication channel. Depending on the implementation, all or a subset of the definitions below is supported.

■ UNKNOWN	#define RCX_COMM_STATE_UNKNOWN	0x00000000
■ NOT_CONFIGURED	#define RCX_COMM_STATE_NOT_CONFIGURED	0x00000001
■ STOP	#define RCX_COMM_STATE_STOP	0x00000002
■ IDLE	#define RCX_COMM_STATE_IDLE	0x00000003
■ OPERATE	#define RCX_COMM_STATE_OPERATE	0x00000004

**Communication Channel Error (All Implementations)**

This field holds the current error code of the communication channel. If the cause of error is resolved, the communication error field is set to zero (= RCX\_SYS\_SUCCESS) again. Not all of the error codes are supported in every implementation. Protocol stacks may use a subset of the error codes below.

■ SUCCESS	#define RCX_SYS_SUCCESS	0x00000000
-----------	-------------------------	------------

**Runtime Failures**

■ WATCHDOG TIMEOUT	#define RCX_E_WATCHDOG_TIMEOUT	0xC000000C
--------------------	--------------------------------	------------

**Initialization Failures**

■ (General) INITIALIZATION FAULT	#define RCX_E_INIT_FAULT	0xC0000100
■ DATABASE ACCESS FAILED	#define RCX_E_DATABASE_ACCESS_FAILED	0xC0000101

**Configuration Failures**

■ NOT CONFIGURED	#define RCX_E_NOT_CONFIGURED	0xC0000119
■ (General) CONFIGURATION FAULT	#define RCX_E_CONFIGURATION_FAULT	0xC0000120
■ INCONSISTENT DATA SET	#define RCX_E_INCONSISTENT_DATA_SET	0xC0000121
■ DATA SET MISMATCH	#define RCX_E_DATA_SET_MISMATCH	0xC0000122
■ INSUFFICIENT LICENSE	#define RCX_E_INSUFFICIENT_LICENSE	0xC0000123
■ PARAMETER ERROR	#define RCX_E_PARAMETER_ERROR	0xC0000124
■ INVALID NETWORK ADDRESS	#define RCX_E_INVALID_NETWORK_ADDRESS	0xC0000125
■ NO SECURITY MEMORY	#define RCX_E_NO_SECURITY_MEMORY	0xC0000126

**Network Failures**

■ (General) NETWORK FAULT	#define RCX_COMM_NETWORK_FAULT	0xC0000140
■ CONNECTION CLOSED	#define RCX_COMM_CONNECTION_CLOSED	0xC0000141
■ CONNECTION TIMED OUT	#define RCX_COMM_CONNECTION_TIMEOUT	0xC0000142
■ LONELY NETWORK	#define RCX_COMM_LONELY_NETWORK	0xC0000143
■ DUPLICATE NODE	#define RCX_COMM_DUPLICATE_NODE	0xC0000144
■ CABLE DISCONNECT	#define RCX_COMM_CABLE_DISCONNECT	0xC0000145

**Version (All Implementations)**

The version field holds version of this structure. It starts with one; zero is not defined.

■ STRUCTURE VERSION	#define RCX_STATUS_BLOCK_VERSION	0x0001
---------------------	----------------------------------	--------

**Watchdog Timeout (All Implementations)**

This field holds the configured watchdog timeout value in milliseconds. The application may set its watchdog trigger interval accordingly. If the application fails to copy the value from the host watchdog location to the device watchdog location, the protocol stack will interrupt all network connections immediately regardless of their current state. For details, see section 4.13 of the netX DPM Interface Manual.

**Host Watchdog (All Implementations)**

The protocol stack supervises the host system using the watchdog function. If the application fails to copy the value from the device watchdog location (section 3.2.5 of the netX DPM Interface Manual) to the host watchdog location (section 3.2.4 of the netX DPM Interface Manual), the protocol stack assumes that the host system has some sort of problem and shuts down all network connections. For details on the watchdog function, refer to section 4.13 of the netX DPM Interface Manual.

**Error Count (All Implementations)**

This field holds the total number of errors detected since power-up, respectively after reset. The protocol stack counts all sorts of errors in this field no matter if they were network related or caused internally.

**Error Log Indicator (All Implementations)**

Not supported yet: The error log indicator field holds the number of entries in the internal error log. If all entries are read from the log, the field is set to zero.

### **3.3.1.2 Master Implementation**

In addition to the common status block as outlined in the previous section, a master firmware maintains the additional structures for the administration of all slaves which are connected to the master. These are not discussed here as they are not relevant for the slave.

### **3.3.1.3 Slave Implementation**

The slave firmware uses only the common structure as outlined in section 3.2.5.1 of the Hilscher netX Dual-Port-Memory Manual.

### 3.3.2 Extended Status

The content of the channel specific extended status block is specific to the implementation. Depending on the protocol, a status area may or may not be present in the dual-port memory. It is always available in the default memory map (see section 3.2.1 of *netX Dual-Port Memory Manual*).

---

**Note:** Have in mind, that 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 NETX_EXTENDED_STATUS_BLOCK_Ttag
{
  UINT8 abExtendedStatus[432];
} NETX_EXTENDED_STATUS_BLOCK_T
```

For the CC-Link Slave protocol implementation, the extended status area is structured as follows:

```
typedef struct CCLINK_SLAVE_EXTENDED_STATE_Ttag
  CCLINK_SLAVE_EXTENDED_STATE_T;

#define CCLINK_SLAVE_EXT_STATE_FLAG_WDG 0x0001L
#define CCLINK_SLAVE_EXT_STATE_CTRL 0x0002L
#define CCLINK_SLAVE_EXT_STATE_NRDY 0x0004L

#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_MSK 0x0001L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK 0x0002L
#define CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK 0x0004L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK 0x0008L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK 0x0010L
#define CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK 0x0060L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK 0x0080L

#define CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK 0x0F00L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK 0xF000L

#define CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK 0x0001L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVALID_NUM_OF_POINTS_MSK 0x0002L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK 0x0004L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK 0x0008L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK 0x0010L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK 0x0020L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK 0x0040L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK 0x0080L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK 0x0100L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK 0x0200L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK 0x0400L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK 0x0800L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK 0x1000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK 0x2000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK 0xC000L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SINGLE_MSK 0xL
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_DOUBLE_MSK 0x4000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_QUADRUPLE_MSK 0x8000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_OCTUPLE_MSK 0xC000L

struct CCLINK_SLAVE_EXTENDED_STATE_Ttag
{
  TLR_UINT32 ulFlags;

  TLR_UINT32 ulMasterState;
  TLR_UINT32 ulSlaveState;

  TLR_UINT32 ulRxByteCount;
  TLR_UINT32 ulRWrByteCount;

  TLR_UINT32 ulRyByteCount;
  TLR_UINT32 ulRWwByteCount;
};
```

The meaning of these parameters is:

**ST1 – Data transfer from master to slave station**

Flag Name	Value	Meaning
CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_MSK	0x01L	Master station (user application program)
CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK	0x02L	Master station user application program error check
CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK	0x04L	Refresh startup
CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK	0x08L	Transient data status
CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK	0x10L	Transient data reception enable
CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK	0x60L	Protocol version
CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK	0x80L	Master station type

Table 16: ST1 – Data Transfer from Master to Slave Station

**Master Station (User Application Program)**

This entry contains the information about the operation status the master station user application program, i.e. whether the application program is running or not. Allowed options are:

Value	Meaning
0	Stop
1	Run

Table 17: Allowed Values of Master Station (User Application Program)

**Master Station User Application Program Error Check**

This entry contains the information whether an error occurred at the master station user application program. Allowed options are:

Value	Meaning
0	Normal
1	Abnormal

Table 18: Allowed Values of Master Station User Application Program Error Check

**Refresh Startup**

This entry contains the information whether link refresh has been started or stopped.

Value	Meaning
0	Stop
1	Start

Table 19: Allowed Values of Refresh Startup

**Transient Data Status**

This entry contains the information about the transient data status, i.e. whether transient data is included.

Value	Meaning
0	Not present
1	Present

Table 20: Allowed Values of Transient Data Status

**Transient Data Reception Enable**

This entry contains the information whether transient data reception is enabled or not.

Value	Meaning
0	Disabled
1	Enabled

Table 21: Allowed Values of Transient Data Reception Enable

**Protocol Version**

This entry contains the protocol version

Value	Meaning
0	Version 1
1	Version 2
2	Version 3 (Future function)
3	Version 4 (Future function)

Table 22: Allowed Values of Protocol Version

**Master Station Type**

This entry contains the information whether the station is an (ordinary) master station or a standby master station.

Value	Meaning
0	Master station
1	Standby master station

*Table 23: Allowed Values of Master Station Type*

**ST1 – Data transfer from Slave to Master Station**

Flag Name	Value	Meaning
CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK	0x01	Fuse status
CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVALID_NUM_OF_POINTS_MSK	0x02	Unit error/invalid No. of points
CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK	0x04	Refresh receive
CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK	0x08	Parameter receive
CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK	0x10	Switch change detection
CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK	0x20	Cyclic transmission flag
CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK	0x40	Reserved
CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK	0x80	WDT error

Table 24: ST1 – Data Transfer from Slave to Master Station

**Fuse Status**

This entry contains the information about the state of the fuse. Allowed values are:

Value	Meaning
0	Normal
1	Abnormal
Data link is continued.	

Table 25: Allowed Values of Fuse Status

**Unit Error/Invalid No. of Points**

This entry contains the information whether

- either a unit error has occurred (in case the slave station is a remote I/O station).
- or the invalid number of points flag has been set (in case the slave station is no remote I/O station)

Allowed values are:

Value	Meaning
0	No
1	Yes <hr/> Data link is aborted. <hr/>

Table 26: Allowed Values of Unit Error/Invalid No. of Points

### Refresh Receive

This entry contains the information whether a refresh has been received. Allowed values are:

Value	Meaning
0	Receive completed
1	Not received <hr/> Data link is aborted. <hr/>

Table 27: Allowed Values of Refresh Receive

### Parameter Receive

This entry contains the information whether a parameter has been received. Allowed values are:

Value	Meaning
0	Receive completed
1	Not received

Table 28: Allowed Values of Parameter Receive

### Switch Change Detection

This entry contains the information whether the position of a switch has been changed. Allowed values are:

Value	Meaning
0	No change
1	A change has occurred <hr/> Data link is continued. <hr/>

Table 29: Allowed Values of Switch Change Detection

**Cyclic Transmission Flag**

This entry contains the information whether cyclic data transmission is active. Allowed values are:

Value	Meaning
0	Cyclic data transmission is enabled
1	Cyclic data transmission is disabled

Table 30: Allowed Values of Cyclic Transmission Flag

**Watchdog Timer Error**

This entry contains the information whether a watchdog timer error has been detected. Allowed values are:

Value	Meaning
0	No watchdog timer error has been detected
1	A watchdog timer error has been detected <hr/> <hr/> Data link is continued.

Table 31: Allowed Values of Watchdog Timer Error

**ST2 – Data Transfer from Master to Slave Station**

Flag Name	Value	Meaning
CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK	0x0F00	RY information transmission points, see below
CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK	0xF000	RW/w information transmission points, see below

Table 32: ST2 – Data Transfer from Master to Slave Station

***RY Information Transmission Points***

The number of RY information transmission points is determined depending on the four lower bits of status byte ST2 according to the following table:

**RY Information Transmission Points**

D3	D2	D1	D0	Number of RY information transmission points	Number of bytes
0	0	0	0	0	0
0	0	0	1	256	32
0	0	1	0	512	64
0	0	1	1	768	96
0	1	0	0	1024	128
0	1	0	1	1280	160
0	1	1	0	1536	192
0	1	1	1	1792	224
1	0	0	0	2048	256
All other combinations (1001 to 1111)				Reserved	

Table 33: RY Information Transmission Points

***RWw Information Transmission Points***

The number of RWw information transmission points is determined depending on the four upper bits of status byte ST2 according to the following table:

**RWw Information Transmission Points**

D3	D2	D1	D0	Number of RWw information transmission points	Number of bytes
0	0	0	0	0	0
0	0	0	1	32	64
0	0	1	0	64	128
0	0	1	1	96	192
0	1	0	0	128	256
0	1	0	1	160	320
0	1	1	0	192	384
0	1	1	1	224	448
1	0	0	0	256	512
All other combinations (1001 to 1111)				Reserved	

Table 34: RWw Information Transmission Points

**ST2 – Data Transfer from Slave to Master Station**

Flag Name	Value	Meaning
CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK	0x01	Transient data status
CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK	0x02	Transient receive
CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK	0x04	Transient type
CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK	0x08	Reserved
CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK	0x10	Transmission status
CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK	0x20	Reserved (set to 1)
CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK	0x40	Extended cycle setting
CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK	0x80	Extended cycle setting

Table 35: ST2 – Data Transfer from Slave to Master Station

**Transient Data Status**

This entry contains the information about the transient data status. Allowed values are:

Value	Meaning
0	Not present
1	Present

Table 36: Allowed Values of Transient Data Status

**Transient Receive**

This entry contains the information whether transient data have been received. Allowed values are:

Value	Meaning
0	Disabled
1	Enabled

Table 37: Allowed Values of Transient Receive

**Transient Type**

This entry contains the information which type of transient data has been selected.

Allowed values are:

Value	Meaning
0	1:n
1	n:n

Table 38: Allowed Values of Transient Type

**Transmission Status**

This entry contains the information about the current transmission status. Allowed values are:

Value	Meaning
0	Normal
1	Abnormal

Table 39: Allowed Values of Transmission Status

**Extended Cycle Setting**

This entry contains the information about the current extended cycle setting. Allowed values are:

Value	Meaning
0x00	Single setting
0x40	Double setting
0x80	Quadruple setting
0xC0	Octuple setting

Table 40: Allowed Values of Extended cycle setting

**ulRxByteCount**

This entry contains the data count of RX data in bytes.

**ulRWrByteCount**

This entry contains the data count of RWr data in bytes

**ulRyByteCount**

This entry contains the data count of RY data in bytes.

***ulRWwByteCount***

This entry contains the data count of RWw data in bytes.

### 3.4 Control Block

A control block is always present within the communication channel. In some respects, control and status block are used together in order to exchange information between host application and netX firmware. The control block is written by the application, whereas the application reads a status block. Both control and status block have registers that use the *Change of State* mechanism (see also section 2.2.1 of the *netX Dual-Port-Memory Manual*.)

The following gives an example of the use of control and status block. The host application wishes to lock the configuration settings of a communication channel to protect them against changes. The application sets the *Lock Configuration* flag in the control block to the communication channel firmware. As a result, the channel firmware sets the *Configuration Locked* flag in the status block (see below), indicating that the current configuration settings cannot be deleted, altered, overwritten or otherwise changed.

The control block of a dual-port memory features a watchdog function to allow the operating system running on the netX supervise the host application and vice versa. The control area is always present in the dual-port memory.

Control Block			
Offset	Type	Name	Description
0x0008	UINT32	ulApplicationCOS	Application Change Of State State Of The Application Program INITIALIZATION, LOCK CONFIGURATION
0x000C	UINT32	ulDeviceWatchdog	Device Watchdog Host System Writes, Protocol Stack Reads

Table 41: Communication Control Block

#### Communication Control Block Structure

```
typedef struct NETX_CONTROL_BLOCK_Ttag
{
  UINT32 ulApplicationCOS;
  UINT32 ulDeviceWatchdog;
} NETX_CONTROL_BLOCK_T;
```

For more information concerning the Control Block please refer to the netX DPM Interface Manual.

## 4 Getting started / Configuration

This section explains some essential information you should know when starting to work with the CC-Link Slave Protocol API.

### 4.1 Overview about Essential Functionality

You can find the most commonly used functionality of the CC-Link Slave Protocol API within the following sections of this document:

Topic	Section Number	Section Name
Configure CC-Link Slave	5.1.2	CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration
Start/Stop network communication	5.2.5	CCLINK_SLAVE_STARTSTOP_REQ/CNF – Start/Stop Communication with Network
Initialization	5.2.1	CCLINK_SLAVE_INITIALIZE_REQ/CNF – Initialization of CC-Link Slave

*Table 42: Overview about Essential Functionality*

## 4.2 Warmstart Parameters

The following table contains relevant information about the warmstart parameters for the CC-Link Slave firmware such as an explanation of the meaning of the parameter and ranges of allowed values:

Parameter	Meaning	Range of Value / Value
System Flags	System flags	<p>BIT 0: AUTOSTART / APPLICATION CONTROLLED communication with a controller after a device start is allowed without BUS_ON flag, but the communication will be interrupted if the BUS_ON flag changes state to 0 communication with controller is allowed only with the BUS_ON flag.</p> <p>BIT 1: I/O STATUS DISABLED/ ENABLED Not supported yet</p> <p>BIT 2: IO STATUS 32 BIT Not supported yet</p> <p>BIT 3 - 31: Reserved for further use, set to zero</p>
CcLink Flags	CC-Link flags	<p>Bit 0: Vendor Code DISABLED/ ENABLED Parameter ulVendorCode will be evaluated if this bit is set. Otherwise the default value will be used</p> <p>Bit 1: Model Type DISABLED/ ENABLED Parameter ulModelType will be evaluated if this bit is set. Otherwise the default value will be used</p> <p>Bit 2: SW Version DISABLED/ ENABLED Parameter ulSwVersion will be evaluated if this bit is set. Otherwise the default value will be used</p> <p>BIT 3 - 31: Reserved for further use, set to zero</p>
Watchdog Time	<p>Watchdog supervision time [ms] within which the device watchdog must be retriggered from the application program while the application program monitoring is activated.</p> <p>Watchdog supervision deactivated Watchdog supervision activated</p>	<p>0 20 .. 65535</p>
Slave Station Address	<p>Station address of CC-Link Slave</p> <p>Warmstart parameter Parameter taken from rotary switch (Allowed for devices with rotary switches)</p> <p><b>Note:</b> The number of occupied stations and station address must not exceed the maximum station address (Address + Number of occupied stations - 1 &lt;= 64).</p>	<p>1 .. 64 255</p>

Baud rate	Network transmission rate <b>Note:</b> Parameter is taken from rotary switch if station address is set to 255 (Allowed for devices with rotary switches).	See Table 44: Available Baud Rate Values
Station type	Type of CC-Link station Remote I/O Station: Remote Device Station:	0 1
Number of occupied stations	Number of occupied stations Remote I/O Station: Remote Device Station: <b>Note:</b> The number of occupied stations and station address must not exceed the maximum station address (Address + Number of occupied stations - 1 <= 64).	1 1 .. 4
CC-Link Version	CC-Link version CC-Link Version 1 CC-Link Version 2	0 1
Extension Cycle	Number of extension cycles Allowed numbers for CC-Link version 1 Single / One cycle Allowed numbers for CC-Link version 2 Single / One cycle Double / Two cycles Quadruple / Four cycles Octuple / Eight cycles	0 0 1 2 3
I/O types/points	I/O types/ total number of I/O points	0..16
HoldClrMode	Behavior in case of bus error Clear output data Hold last received output data	0 1
Vendor Code	Vendor code (If corresponding bit in parameter <code>ulCcLinkFlags</code> parameter is set)	0 . 65535
Model Type	Model type (If corresponding bit in parameter <code>ulCcLinkFlags</code> parameter is set)	0 .. 255
Sw Version	Software version (If corresponding bit in parameter <code>ulCcLinkFlags</code> parameter is set)	0 .. 63

Table 43: Meaning and allowed Values for Warmstart-Parameters.

The applicable baud rates can be coded with the values given in the following table:

Baud rate	Symbolic Constant	Value
156 kBaud	CCLINK_SLAVE_BAUD_156K	0
625 kBaud	CCLINK_SLAVE_BAUD_625K	1
2500 kBaud	CCLINK_SLAVE_BAUD_2500K	2
5 MBaud	CCLINK_SLAVE_BAUD_5M	3
10 MBaud	CCLINK_SLAVE_BAUD_10M	4

Table 44: Available Baud Rate Values

#### 4.2.1 Behavior when receiving a Set Configuration / Warmstart Command

The following rules apply for the behavior of the CC-Link Slave protocol stack when receiving a set configuration command:

- The configuration packets name is CCLINK\_APS\_SET\_CONFIGURATION\_REQ for the request and CCLINK\_APS\_SET\_CONFIGURATION\_CNF for the confirmation.
- The configuration data are checked for consistency and integrity.
- In case of failure no data are accepted.
- In case of success the configuration parameters are stored internally (within the RAM).
- The parameterized data will be activated only after a channel init has been performed.
- No automatic registration of the application at the stack happens.
- The confirmation packet CCLINK\_APS\_SET\_CONFIGURATION\_CNF only transfers simple status information, but does not repeat the whole parameter set.

For all versions up to firmware version V2.1.8.0, only the warmstart command (the predecessor of the set configuration command) was present showing up the following deviations from the behavior described above:

1. Contrary to the situation when receiving a set configuration command, on every received warmstart packet an automatic channel-init is performed..
2. The entire parameter set is completely delivered within the CCLINK\_APS\_SET\_CONFIGURATION\_CNF or CCLINK\_APS\_WARMSTART\_CNF packet.

## 4.3 Input and Output Data

Depending on the CC-Link Slave configuration, the input and output data area is subdivided into different sections:

### 4.3.1 Input and Output Data for CC-Link Version 1

- Input and Output Data for Remote I/O Station

I/O Offset	Area	Length (Byte)	Type
0	Output block	4	RX
0	Input block	4	RY

Table 45: Input and Output Data for Remote I/O Device

- Input and Output Data for Remote Device Station with One Occupied Station

I/O Offset	Area	Length (Byte)	Type
0	Output block	4	RX
4	Output block	8	RWr
0	Input block	4	RY
4	Input block	8	RWw

Table 46: Input and Output Data for Remote Device Station with One Occupied Station

- Input and Output Data for Remote Device Station with Two Occupied Stations

I/O Offset	Area	Length (Byte)	Type
0	Output block	8	RX
8	Output block	16	RWr
0	Input block	8	RY
8	Input block	16	RWw

Table 47: Input and Output Data for Remote Device Station with Two Occupied Stations

- Input and Output Data for Remote Device Station with Three Occupied Stations

I/O Offset	Area	Length (Byte)	Type
0	Output block	12	RX
12	Output block	24	RWr
0	Input block	12	RY
12	Input block	24	RWw

Table 48: Input and Output Data for Remote Device Station with Three Occupied Stations

## ■ Input and Output Data for Remote Device Station with Four Occupied Stations

I/O Offset	Area	Length (Byte)	Type
0	Output block	16	RX
16	Output block	32	RWr
0	Input block	16	RY
16	Input block	32	RWw

*Table 49: Input and Output Data for Remote Device Station with Four Occupied Stations*

### 4.3.2 Input and Output Data for CC-Link Version 2

Depending on the CC-Link Slave configuration, the input and output data area is subdivided into different sections:

- Input and Output Data for Remote Device Station with One Occupied Station, Single Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	4	RX
4	Output block	8	RWr
0	Input block	4	RY
4	Input block	8	RWw

Table 50: Input and Output Data for Remote Device Station with One Occupied Stations, Single Setting

- Input and Output Data for Remote Device Station with Two Occupied Stations, Single Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	8	RX
8	Output block	16	RWr
0	Input block	8	RY
8	Input block	16	RWw

Table 51: Input and Output Data for Remote Device Station with Two Occupied Stations, Single Setting

- Input and Output Data for Remote Device Station with Three Occupied Stations, Single Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	12	RX
12	Output block	24	RWr
0	Input block	12	RY
12	Input block	24	RWw

Table 52: Input and Output Data for Remote Device Station with Three Occupied Stations, Single Setting

- Input and Output Data for Remote Device Station with Four Occupied Stations, Single Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	16	RX
16	Output block	32	RWr
0	Input block	16	RY
16	Input block	32	RWw

Table 53: Input and Output Data for Remote Device Station with Four Occupied Stations, Single Setting

■ Input and Output Data for Remote Device Station with One Occupied Station, Double Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	4	RX
4	Output block	16	RWr
0	Input block	4	RY
4	Input block	16	RWw

Table 54: Input and Output Data for Remote Device Station with One Occupied Station, Double Setting

■ Input and Output Data for Remote Device Station with Two Occupied Stations, Double Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	12	RX
12	Output block	32	RWr
0	Input block	12	RY
12	Input block	32	RWw

Table 55: Input and Output Data for Remote Device Station with Two Occupied Stations, Double Setting

■ Input and Output Data for Remote Device Station with Three Occupied Stations, Double Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	20	RX
20	Output block	48	RWr
0	Input block	20	RY
20	Input block	48	RWw

Table 56: Input and Output Data for Remote Device Station with Three Occupied Stations, Double Setting

■ Input and Output Data for Remote Device Station with Four Occupied Stations, Double Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	28	RX
28	Output block	64	RWr
0	Input block	28	RY
28	Input block	64	RWw

Table 57: Input and Output Data for Remote Device Station with Four Occupied Stations, Double Setting

- Input and Output Data for Remote Device Station with One Occupied Station, Quadruple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	8	RX
8	Output block	32	RWr
0	Input block	8	RY
8	Input block	32	RWw

Table 58: Input and Output Data for Remote Device Station with One Occupied Station, Quadruple Setting

- Input and Output Data for Remote Device Station with Two Occupied Stations, Quadruple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	24	RX
24	Output block	64	RWr
0	Input block	24	RY
24	Input block	64	RWw

Table 59: Input and Output Data for Remote Device Station with Two Occupied Stations, Quadruple Setting

- Input and Output Data for Remote Device Station with Three Occupied Stations, Quadruple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	40	RX
40	Output block	96	RWr
0	Input block	40	RY
40	Input block	96	RWw

Table 60: Input and Output Data for Remote Device Station with Three Occupied Stations, Quadruple Setting

- Input and Output Data for Remote Device Station with Four Occupied Stations, Quadruple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	56	RX
56	Output block	128	RWr
0	Input block	56	RY
56	Input block	128	RWw

Table 61: Input and Output Data for Remote Device Station with Four Occupied Stations, Quadruple Setting

■ Input and Output Data for Remote Device Station with One Occupied Station, Octuple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	16	RX
16	Output block	64	RWr
0	Input block	16	RY
16	Input block	64	RWw

Table 62: Input and Output Data for Remote Device Station with One Occupied Station, Octuple Setting

■ Input and Output Data for Remote Device Station with Two Occupied Stations, Octuple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	48	RX
48	Output block	128	RWr
0	Input block	48	RY
48	Input block	128	RWw

Table 63: Input and Output Data for Remote Device Station with Two Occupied Stations, Octuple Setting

■ Input and Output Data for Remote Device Station with Three Occupied Stations, Octuple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	80	RX
80	Output block	192	RWr
0	Input block	80	RY
80	Input block	192	RWw

Table 64: Input and Output Data for Remote Device Station with Three Occupied Stations, Octuple Setting

■ Input and Output Data for Remote Device Station with Four Occupied Stations, Octuple Setting

I/O Offset	Area	Length (Byte)	Type
0	Output block	112	RX
112	Output block	256	RWr
0	Input block	112	RY
112	Input block	256	RWw

Table 65: Input and Output Data for Remote Device Station with Four Occupied Stations, Octuple Setting

## 4.4 Task Structure of the CC-Link Slave Stack

The illustration below displays the internal structure of the tasks which together represent the CC-Link Slave Stack:

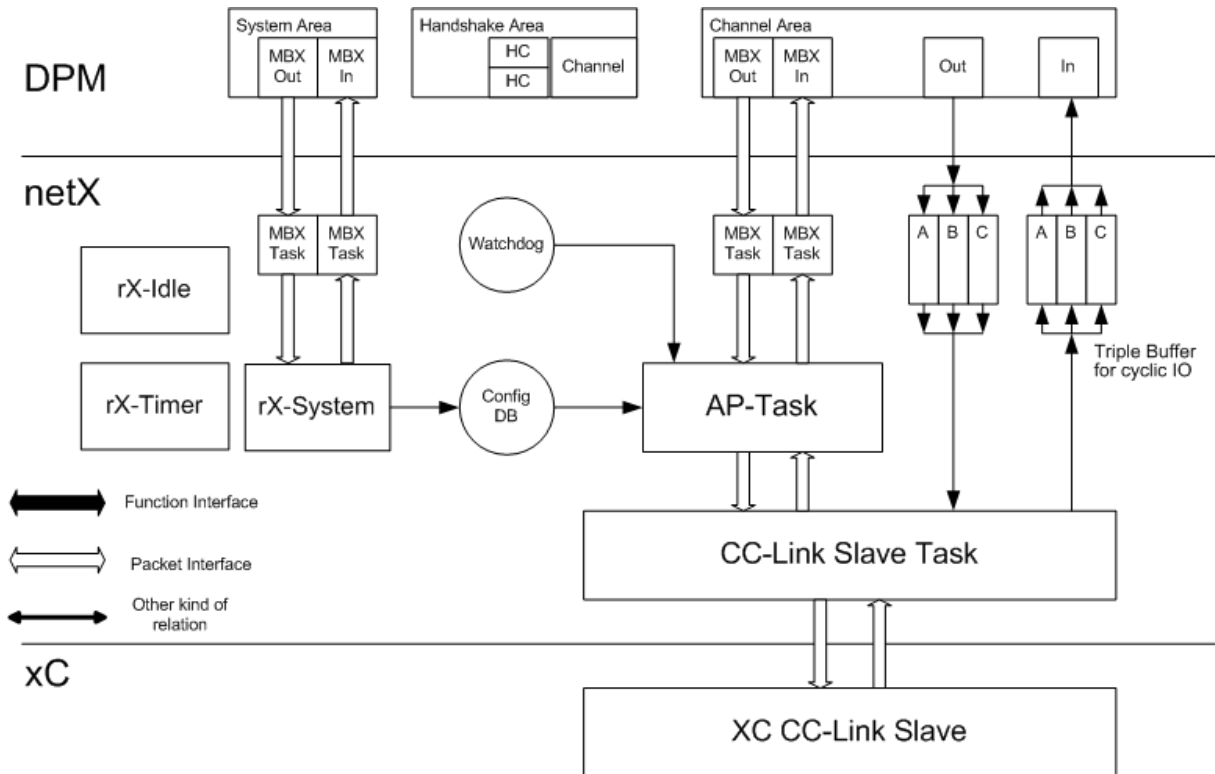


Figure 6: Internal Structure of CC-Link Slave Firmware

For the explanation of the different kinds of arrows see lower left corner of figure.

The dual-port memory is used for exchange of information, data and packets. Configuration and IO data will be transferred using this way.

The user application only accesses the task located in the highest layer namely the AP task which constitutes the application interface of the CC-Link Slave stack.

The triple buffer mechanism provides a consistent synchronous access procedure from both sides (DPM and AP task). The triple buffer technique ensures that the access will always affect the last written cell.

In detail, the various tasks have the following functionality and responsibilities:

### **AP-Task**

The AP-Task provides the interface to the user application and the control of the stack. It also completely handles the DualPort Memory interface of the communication channel. In detail, it is responsible for the following:

- - Handling the communication channels DPM-interface
- - Configuration of protocol stack
- - IO Process data exchange
- - Channel mailboxes
- - Watchdog supervision
- - Handling of applications packets
- - Send/Receive packets

### **CC-Link Slave Task**

The CC-Link Slave Task is the CC-Link Slave stack implementation. It is responsible for the protocol handling, the communication to/from XC layer and it is the counterpart of the AP-Task.

## 5 The Application Interface

This chapter defines the application interface of the CC-Link Slave stack.

The application itself has to be developed as a task according to the Hilscher's Task Layer Reference Model. The application task is named AP-Task in the following sections and chapters.

The AP-Task's process queue shall keep track of its incoming packets. It provides the communication channel for the underlying CC-Link Slave Stack. Once, the CC-Link Slave stack communication is established, events received by the stack are mapped to packets that are sent to the AP-Task's process queue. Every packet has to be evaluated in the AP-Task's context and corresponding actions be executed. Additionally, Initiator-Services that are to be requested by the application are sent via predefined queue macros to the underlying CC-Link Slave stack queues via packets as well.

The following chapters describe the packets that may be received or sent by the AP-Task.

## 5.1 The CC-Link APS-Task

The CC-Link APS-Task coordinates, within the CC-Link Slave stack, the overlaying functions.

It is responsible for all application interactions and represents the counterpart of the AP-Task within the existing CC-Link Slave stack implementation.

To get the handle of the process queue of the CC-Link APS-Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII Queue name	Description
"QUE_CCLAPS"	Name of the CC-Link APS-Task process queue

Table 66: CC-Link APS-Task Process Queue

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the CC-Link APS -Task. This handle is the same handle that has to be used in conjunction with the macros `TLR_QUE_SENDFPACKET_FIFO/LIFO()` for sending a packet to the CC-Link APS-Task.

In detail, the following functionality is provided by the APS-Task:

Overview over Packets of the CC-Link APS-Task			
No. of section	Packet	Command code (REQ/CNF or IND/RES)	Page
5.1.1	CCLINK_APS_WARMSTART_REQ/CNF – Set Warmstart Parameters	0x4600/ 0x4601	66
5.1.2	CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration	0x4604/ 0x4605	76

Table 67: Overview over the Packets of the APS-Task of the CC-Link Slave Protocol Stack

### 5.1.1 CCLINK\_APS\_WARMSTART\_REQ/CNF – Set Warmstart Parameters

This service can be used by the user application in order to configure the AP-task with warmstart parameters. After this request is received, the AP-task will configure the CC-Link Slave task with the given parameters from this request and, if configured, starts the communication with the CC-Link network. This request will be denied if the configuration lock flag is set (for more information on this topic see section “*All Implementations*”).



**Note:** The packet described in this section is obsolete and is no longer supported. Do not use this packet for all new developments! It is replaced by the packet *CCLINK\_APS\_SET\_CONFIGURATION\_REQ/CNF – Set Configuration* described in the next section and has to be used for new developments.

#### Packet Structure Reference

```
typedef struct CCLINK_APS_WARMSTART_REQ_DATA_Ttag
    CCLINK_APS_WARMSTART_REQ_DATA_T;

#define CCLINK_APS_SYS_FLAG_COM_CONTROLLED_RELEASE    0x0001L
#define CCLINK_APS_SYS_FLAG_IO_STATUS_ENABLED        0x0002L
#define CCLINK_APS_SYS_FLAG_IO_STATUS_32_BIT         0x0004L

#define CCLINK_APS_WD_OFF                             0xL
#define CCLINK_APS_WD_MIN_TIMEOUT                    0x0014L
#define CCLINK_APS_WD_MAX_TIMEOUT                    0xFFFFL

#define CCLINK_APS_CCLS_FLAGS_CFG_VENDOR_CODE        0x0001L
#define CCLINK_APS_CCLS_FLAGS_CFG_MODEL_TYPE        0x0002L
#define CCLINK_APS_CCLS_FLAGS_CFG_SW_VERSION        0x0004L

#define CCLINK_APS_STATION_ADDR_CFG_ROTARY_SWITCH    0x00FFL

struct CCLINK_APS_WARMSTART_REQ_DATA_Ttag
{
    TLR_UINT32    ulSystemFlags;
    TLR_UINT32    ulCcLinkFlags;

    TLR_UINT32    ulWdgTime;

    TLR_UINT32    ulSlaveStationAddr;
    TLR_UINT32    ulBaudRate;

    TLR_UINT32    ulStationType;
    TLR_UINT32    ulNoStation;
    TLR_UINT32    ulCcLinkVersion;
    TLR_UINT32    ulExtensionCycle;
    TLR_UINT32    ulReserved;
    TLR_BOOLEAN32 fHoldClrMode;

    TLR_UINT32    ulVendorCode;
    TLR_UINT32    ulModelType;
    TLR_UINT32    ulSwVersion;
};
```

```
typedef struct CCLINK_APS_PCK_WARMSTART_REQ_Ttag
    CCLINK_APS_PCK_WARMSTART_REQ_T;

struct CCLINK_APS_PCK_WARMSTART_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_APS_WARMSTART_REQ_DATA_T tData;
}
```

**Packet Description**

structure CCLINK_APS_PCK_WARMSTART_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLAPS	Destination Queue-Handle
	ulSrc	UINT32	0 ... $2^{32}-1$	Source Queue-Handle
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	56	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		<i>See section 6.1 Status/Error Codes CC-Link APS-Task</i>
	ulCmd	UINT32	0x4600	CCLINK_APS_WARMSTART_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

tData	structure CCLINK_APS_WARMSTART_REQ_DATA_T			
	ulSystemFlags	UINT32	<p>0</p> <p>1</p>	<p>System Flags</p> <p>BIT 0: AUTOSTART / APPLICATION CONTROLLED</p> <p>Communication with a controller after a device start is allowed without BUS_ON flag, but the communication will be interrupted if the BUS_ON flag changes state to 0.</p> <p>communication with controller is allowed only with the BUS_ON flag.</p> <p>BIT 1: I/O STATUS DISABLED/ ENABLED</p> <p>Not supported yet</p> <p>BIT 2: IO STATUS 32 BIT</p> <p>Not supported yet</p> <p>BIT 3 - 31: Reserved for further use, set to zero</p>
	ulCcLinkFlags	UINT32	<p>0</p> <p>1</p> <p>0</p> <p>1</p> <p>0</p> <p>1</p>	<p>CC-Link Flags</p> <p>Bit 0: CONFIG VENDOR CODE DISABLED/ENABLED</p> <p>Default Vendor Code is used Value from parameter 'ulVendorCode' is used</p> <p>Bit 1: CONFIG MODEL TYPE DISABLED/ENABLED</p> <p>Default Model Type is used Value from parameter 'ulModelType' is used</p> <p>Bit 2: CONFIG SW VERSION DISABLED/ENABLED</p> <p>Default Software Version is used Value from parameter 'ulSwVersion' is used</p>
	ulWdgTime	UINT32	<p>0</p> <p>20 .. 65535</p>	<p>Watchdog supervision</p> <p>Watchdog supervision deactivated</p> <p>Watchdog time in milliseconds</p>
	ulSlaveStationAddr	UINT32	<p>1 .. 64</p> <p>255</p>	<p>Slave Station Address</p> <p>Slave Station Address from Packet</p> <p>Slave Station Address and Baudrate from rotary switches (On devices with rotary switches, only)</p>
	ulBaudRate	UINT32	<p>0</p> <p>1</p> <p>2</p> <p>3</p> <p>4</p>	<p>Baudrate</p> <p>156 kBaud</p> <p>625 kBaud</p> <p>2500 kBaud</p> <p>5 MBaud</p> <p>10 MBaud</p> <p><b>Note:</b> Parameter will not be used if rotary switches are used for configuration. Parameter ulSlaveStationAddr has to be set to 255 in order to activate rotary switch configuration (On devices with rotary switches, only)</p>
	ulStationType	UINT32	<p>0</p> <p>1</p>	<p>Station Type</p> <p>Remote I/O Station</p> <p>Remote Device Station</p>

	ulNoStation	UINT32	1 1 .. 4	Number of Occupied Stations Range for Remote I/O Station Range for Remote Device Station
	ulCcLinkVersion	UINT32	0 1	CC-Link Version Version 1 Version 2 (Remote Device Station only)
	ulExtensionCycle	UINT32	0  0 1 2 3	Number of extension cycles for CC-Link Version 1 Single setting / one cycle Number of extension cycles for CC-Link Version 2 Single setting / one cycle Double setting / two cycles Quadruple setting / four cycles Octuple setting / eight cycles
	ulReserved	UINT32	0	Reserved for further use, set to zero
	fHoldClrMode	BOOLEAN32	0 1	Behavior in case of bus error Clear output data Hold last received output data
	ulVendorCode	UINT32	0 .. 65535	Vendor code (If corresponding bit in parameter ulCcLinkFlags parameter is set)
	ulModelType	UINT32	0 .. 255	Model type (If corresponding bit in parameter ulCcLinkFlags is set)
	ulSwVersion	UINT32	0 .. 63	Software version (If corresponding bit in parameter ulCcLinkFlags is set)

Table 68: CCLINK\_APS\_PCK\_WARMSTART\_REQ\_T – Set Warmstart Parameter Request

**Packet Structure Reference**

```
typedef struct CCLINK_APS_WARMSTART_CNF_DATA_Ttag
    CCLINK_APS_WARMSTART_CNF_DATA_T;

struct CCLINK_APS_WARMSTART_CNF_DATA_Ttag
{
    TLR_UINT32      ulSystemFlags;
    TLR_UINT32      ulCcLinkFlags;

    TLR_UINT32      ulWdgTime;

    TLR_UINT32      ulSlaveStationAddr;
    TLR_UINT32      ulBaudRate;

    TLR_UINT32      ulStationType;

    TLR_UINT32      ulNoStation;

    TLR_UINT32      ulCcLinkVersion;
    TLR_UINT32      ulExtensionCycle;

    TLR_UINT32      ulReserved;
    TLR_BOOLEAN32   fHoldClrMode;

    TLR_UINT32      ulVendorCode;
    TLR_UINT32      ulModelType;
    TLR_UINT32      ulSwVersion;
};

typedef struct CCLINK_APS_PCK_WARMSTART_CNF_Ttag
    CCLINK_APS_PCK_WARMSTART_CNF_T;

struct CCLINK_APS_PCK_WARMSTART_CNF_Ttag
{
    TLR_PACKET_HEADER_T      tHead;
    CCLINK_APS_WARMSTART_CNF_DATA_T tData;
};
```

**Packet Description**

structure CCLINK_APS_PCK_WARMSTART_CNF_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	56	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.1 Status/Error Codes CC-Link APS-Task
	ulCmd	UINT32	0x4601	CCLINK_APS_WARMSTART_CNF - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch



	ulNoStation	UINT32	1 1 .. 4	Number of Occupied Stations Range for Remote I/O Station Range for Remote Device Station
	ulCcLinkVersion	UINT32	0 1	CC-Link Version Version 1 Version 2 (Remote Device Station only)
	ulExtensionCycle	UINT32	0  0 1 2 3	Number of extension cycles for CC-Link Version 1 Single setting / one cycle Number of extension cycles for CC-Link Version 2 Single setting / one cycle Double setting / two cycles Quadruple setting / four cycles Octuple setting / eight cycles
	ulReserved	UINT32	0	Reserved for further use, set to zero
	fHoldClrMode	BOOLEAN32	0 1	Behavior in case of bus error Clear output data Hold last received output data
	ulVendorCode	UINT32	0 .. 65535	Vendor code (If corresponding bit in parameter ulCcLinkFlags parameter is set)
	ulModelType	UINT32	0 .. 255	Model type (If corresponding bit in parameter ulCcLinkFlags is set)
	ulSwVersion	UINT32	0 .. 63	Software version (If corresponding bit in parameter ulCcLinkFlags is set)

*Table 69: CCLINK\_APS\_PCK\_WARMSTART\_CNF\_T – Set Warmstart Parameter Confirmation*

**Packet Status/Error**

Definition / (Value)	Description
TLR_S_OK (0x)	Status ok
TLR_I_CCLINK_APS_CONFIG_LOCK (0x406B0009)	Configuration is locked.
TLR_E_CCLINK_APS_PACKET_LENGTH (0xC06B0008)	Invalid packet length.
TLR_E_CCLINK_APS_STATION_TYPE_PARAMETER (0xC06B001A)	Invalid parameter for station type.
TLR_E_CCLINK_APS_NO_STATION_PARAMETER (0xC06B000D)	Invalid parameter for number of stations.
TLR_E_CCLINK_APS_STATION_ADDR_WITH_NO_STATIONS_PARAMETER (0xC06B001B)	Invalid parameter for station address in combination with number of stations.
TLR_E_CCLINK_APS_CCLINK_VERSION_PARAMETER (0xC06B0019)	Invalid parameter for CC-Link version.
TLR_E_CCLINK_APS_VENDOR_CODE_PARAMETER (0xC06B000F)	Invalid parameter for vendor code.
TLR_E_CCLINK_APS_MODEL_TYPE_PARAMETER (0xC06B0012)	Invalid parameter for model type.
TLR_E_CCLINK_APS_SW_VERSION_PARAMETER (0xC06B0011)	Invalid parameter for software version.
TLR_E_CCLINK_APS_DATABASE_FOUND (0xC06B000A)	Configuration database found.
TLR_E_CCLINK_APS_REQUEST_RUNNING (0xC06B0014)	Request already running.
TLR_E_CCLINK_APS_EXTENSION_CYCLE_PARAMETER (0xC06B001C)	Invalid parameter extension cycle.
TLR_E_CCLINK_APS_STATION_TYPE_WITH_CCLINK_VERSION_PARAMETER (0xC06B001D)	Invalid parameter for station type in combination with CC-Link version.

**Table 70:** CCLINK\_APS\_PCK\_WARMSTART\_CNF – Packet Status/Error

### 5.1.2 CCLINK\_APS\_SET\_CONFIGURATION\_REQ/CNF – Set Configuration

This service can be used by the user application in order to configure the AP-task with warmstart parameters. After this request is received, the AP-task will configure the CC-Link Slave task with the given parameters from this request and, if configured, starts the communication with the CC-Link network. The following applies:

- Configuration parameters will be stored internally.
- In case of any error no data will be stored at all.
- A channel init is required to activate the parameterized data.
- This packet does not perform any registration at the stack automatically. Registering must be performed with a separate packet such as the registration packet described in the netX Dual-Port-Memory Manual (RCX\_REGISTER\_APP\_REQ, code 0x2F10).
- This request will be denied if the configuration lock flag is set  
(for more information on this topic see section “All Implementations”).

The parameter `ulIoTypesPoints` can be used to adjust:

the total number of I/O points

the I/O types

The total number of I/O points can have the following values:

*Dependent on the number of occupied stations*

*8 points*

*16 points*

*32 points*

The following I/O types are available:

*Mixed*

*Input*

*Output*

*Composite*

Mixed means the situation when both input and output exist on the same module. The same I/O numbers are used, see CC-Link specification.

Composite means a device that doesn't use the same numbers for input and output.

The following values are possible for parameter `ulIoTypesPoints`:

Code	Value	total number of I/O points	I/O type
CCLINK_SLAVE_IO_TYPES_POINTS_DEFAULT	0x00000000L	Use this value to work with default settings	
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_DEP_ON_STATION	0x00000001L	*	<i>Mixed</i>
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_8POINTS	0x00000002L	8	<i>Mixed</i>
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_16POINTS	0x00000003L	16	<i>Mixed</i>
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_32POINTS	0x00000004L	32	<i>Mixed</i>
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_DEP_ON_STATION	0x00000005L	*	<i>Input</i>
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_8POINTS	0x00000006L	8	<i>Input</i>
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_16POINTS	0x00000007L	16	<i>Input</i>
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_32POINTS	0x00000008L	32	<i>Input</i>
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_DEP_ON_STATION	0x00000009L	*	<i>Output</i>
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_8POINTS	0x0000000AL	8	<i>Output</i>
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_16POINTS	0x0000000BL	16	<i>Output</i>
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_32POINTS	0x0000000CL	32	<i>Output</i>
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_DEP_ON_STATION	0x0000000DL	*	<i>Composite</i>
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_8POINTS	0x0000000EL	8	<i>Composite</i>
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_16POINTS	0x0000000FL	16	<i>Composite</i>
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_32POINTS	0x00000010L	32	<i>Composite</i>

Table 71: Possible Values for Parameter `ulIoTypesPoints`

\* means the total number of I/O points depends on the number of occupied stations.

**Packet Structure Reference**

```

typedef struct CCLINK_APS_SET_CONFIGURATION_REQ_DATA_Ttag
    CCLINK_APS_SET_CONFIGURATION_REQ_DATA_T;

#define CCLINK_APS_SYS_FLAG_COM_CONTROLLED_RELEASE    0x0001L
#define CCLINK_APS_SYS_FLAG_IO_STATUS_ENABLED        0x0002L
#define CCLINK_APS_SYS_FLAG_IO_STATUS_32_BIT         0x0004L

#define CCLINK_APS_WD_OFF                             0xL
#define CCLINK_APS_WD_MIN_TIMEOUT                     0x0014L
#define CCLINK_APS_WD_MAX_TIMEOUT                     0xFFFFL

#define CCLINK_APS_CCLS_FLAGS_CFG_VENDOR_CODE        0x0001L
#define CCLINK_APS_CCLS_FLAGS_CFG_MODEL_TYPE        0x0002L
#define CCLINK_APS_CCLS_FLAGS_CFG_SW_VERSION        0x0004L

#define CCLINK_APS_STATION_ADDR_CFG_ROTARY_SWITCH    0x00FFL

struct CCLINK_APS_SET_CONFIGURATION_REQ_DATA_Ttag
{
    TLR_UINT32    ulSystemFlags;
    TLR_UINT32    ulWdgTime;
    TLR_UINT32    ulCcLinkFlags;

    TLR_UINT32    ulSlaveStationAddr;
    TLR_UINT32    ulBaudRate;

    TLR_UINT32    ulStationType;
    TLR_UINT32    ulNoStation;
    TLR_UINT32    ulCcLinkVersion;
    TLR_UINT32    ulExtensionCycle;
    TLR_UINT32    ulIoTypesPoints;
    TLR_BOOLEAN32 fHoldClrMode;

    TLR_UINT32    ulVendorCode;
    TLR_UINT32    ulModelType;
    TLR_UINT32    ulSwVersion;
};

typedef struct CCLINK_APS_PCK_SET_CONFIGURATION_REQ_Ttag
    CCLINK_APS_PCK_SET_CONFIGURATION_REQ_T;

struct CCLINK_APS_PCK_SET_CONFIGURATION_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead;
    CCLINK_APS_SET_CONFIGURATION_REQ_DATA_T    tData;
}

```

**Packet Description**

structure CCLINK_APS_PCK_SET_CONFIGURATION_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLAPS	Destination Queue-Handle
	ulSrc	UINT32	0 ... $2^{32}-1$	Source Queue-Handle
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	56	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.1 Status/Error Codes CC-Link APS-Task
	ulCmd	UINT32	0x4604	CCLINK_APS_SET_CONFIGURATION_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure CCLINK_APS_SET_CONFIGURATION_REQ_DATA_T			
	ulSystemFlags	UINT32	0 1	System Flags  BIT 0: AUTOSTART / APPLICATION CONTROLLED Communication with a controller after a device start is allowed without BUS_ON flag, but the communication will be interrupted if the BUS_ON flag changes state to 0. communication with controller is allowed only with the BUS_ON flag.  BIT 1: I/O STATUS DISABLED/ ENABLED Not supported yet  BIT 2: IO STATUS 32 BIT Not supported yet  BIT 3 - 31: Reserved for further use, set to zero
	ulWdgTime	UINT32	0 20 .. 65535	Watchdog supervision Watchdog supervision deactivated Watchdog time in milliseconds
	ulCcLinkFlags	UINT32	0 1	CC-Link Flags  Bit 0: CONFIG VENDOR CODE DISABLED/ENABLED Default Vendor Code is used Value from parameter 'ulVendorCode' is used

			0 1	Bit 1: CONFIG MODEL TYPE DISABLED/ENABLED  Default Model Type is used Value from parameter 'ulModelType' is used
			0 1	Bit 2: CONFIG SW VERSION DISABLED/ENABLED  Default Software Version is used Value from parameter 'ulSwVersion' is used
	ulSlaveStationAddr	UINT32	1 .. 64 255	Slave Station Address  Slave Station Address from Packet Slave Station Address and Baudrate from rotary switches (On devices with rotary switches, only)
	ulBaudRate	UINT32	0 1 2 3 4	Baudrate  156 kBaud 625 kBaud 2500 kBaud 5 MBaud 10 MBaud  <b>Note:</b> Parameter will not be used if rotary switches are used for configuration. Parameter ulSlaveStationAddr has to be set to 255 in order to activate rotary switch configuration (On devices with rotary switches, only)
	ulStationType	UINT32	0 1	Station Type  Remote I/O Station Remote Device Station
	ulNoStation	UINT32	1 1 .. 4	Number of Occupied Stations  Range for Remote I/O Station Range for Remote Device Station
	ulCcLinkVersion	UINT32	0 1	CC-Link Version  Version 1 Version 2 (Remote Device Station only)
	ulExtensionCycle	UINT32	0  0 1 2 3	Number of extension cycles for CC-Link Version 1 Single setting / one cycle  Number of extension cycles for CC-Link Version 2  Single setting / one cycle Double setting / two cycles Quadruple setting / four cycles Octuple setting / eight cycles
	ulIoTypesPoints	UINT32	0..16	I/O types and points, see detailed explanation on top of this subsection
	fHoldClrMode	BOOLEAN32	0 1	Behavior in case of bus error  Clear output data Hold last received output data
	ulVendorCode	UINT32	0 .. 65535	Vendor code (If corresponding bit in parameter ulCcLinkFlags parameter is set)
	ulModelType	UINT32	0 .. 255	Model type (If corresponding bit in parameter ulCcLinkFlags is set)
	ulSwVersion	UINT32	0 .. 63	Software version (If corresponding bit in parameter ulCcLinkFlags is set)

Table 72: CCLINK\_APS\_PCK\_SET\_CONFIGURATION\_REQ\_T – Set Warmstart Parameter Request

**Packet Structure Reference**

```

typedef struct CCLINK_APS_PCK_SET_CONFIGURATION_CNF_Ttag
  CCLINK_APS_PCK_SET_CONFIGURATION_CNF_T;

struct CCLINK_APS_PCK_SET_CONFIGURATION_CNF_Ttag
{
  TLR_PACKET_HEADER_T          tHead;
};

```

**Packet Description**

structure CCLINK_APS_PCK_SET_CONFIGURATION_CNF_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.1 Status/Error Codes CC-Link APS-Task
	ulCmd	UINT32	0x4605	CCLINK_APS_SET_CONFIGURATION_CNF - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

*Table 73: CCLINK\_APS\_PCK\_SET\_CONFIGURATION\_CNF\_T – Set Warmstart Parameter Confirmation*

## 5.2 The CC-Link Slave-Task

The CC-Link Slave-Task handles, within the CC-Link Slave stack, the CC-Link protocol.

To get the handle of the process queue of the CC-Link Slave-Task-Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII Queue name	Description
"QUE_CCLSLAVE"	Name of the CC-Link Slave-Task process queue

*Table 74: CC-Link APS-Task Process Queue*

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the CC-Link Slave-Task. This handle is the same handle that has to be used in conjunction with the macros `TLR_QUE_SENDBUFFER_FIFO/LIFO()` for sending a packet to the CC-Link Slave-Task.

In detail, the following functionality is provided by the CC-Link Slave-Task:

<b>Overview over Packets of the CC-Link Slave-Task</b>			
<b>No. of section</b>	<b>Packet</b>	<b>Command code (REQ/CNF or IND/RES)</b>	<b>Page</b>
5.2.1	CCLINK_SLAVE_INITIALIZE_REQ/CNF – Initialization of CC-Link Slave	0x4500 / 0x4501	84
5.2.2	CCLINK_SLAVE_REGISTER_REQ/CNF – Register Application	0x4502 / 0x4503	87
5.2.3	CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle	0x4504 / 0x4505	91
5.2.4	CCLINK_SLAVE_SET_BUSPARAM_REQ/CNF – Set Bus Parameters	0x4506 / 0x4507	95
5.2.5	CCLINK_SLAVE_STARTSTOP_REQ/CNF – Start/Stop Communication with Network	0x4508 / 0x4509	101
5.2.6	CCLINK_SLAVE_GET_CCL_STATUS_REQ/CNF – Get CC-Link Status	0x450A / 0x450B	105
5.2.7	CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ/CNF – Change CC-Link Slave Status	0x450C / 0x450D	109
5.2.8	CCLINK_SLAVE_GET_BUS_PARAM_REQ/CNF – Get Bus Parameters	0x450E / 0x450F	115
5.2.9	CCLINK_SLAVE_STATE_CHANGE_IND/RES – Change of State Indication	0x451E / 0x451F	118
5.2.10	CCLINK_SLAVE_SET_WATCHDOG_FAIL_REQ/CNF – Set Watchdog Fail	0x45AA / 0x45AB	123

Table 75: Overview over the Packets of the CC-Link Slave-Task k of the CC-Link Slave Protocol Stack

### 5.2.1 CCLINK\_SLAVE\_INITIALIZE\_REQ/CNF – Initialization of CC-Link Slave

This request is used in order to reset the CC-Link Slave. This request will be denied if the configuration lock flag is set (for more information on this topic see section 3.3.1.1 “All Implementations”).



**Note:** This command does not delete configuration databases. If the CC-Link Slave is configured by configuration database, this configuration is reloaded again after the initialize command is completed.

#### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_INITIALIZE_REQ_DATA_Ttag
    CCLINK_SLAVE_INITIALIZE_REQ_DATA_T;

struct CCLINK_SLAVE_INITIALIZE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

typedef struct CCLINK_SLAVE_PACKET_INITIALIZE_REQ_Ttag
    CCLINK_SLAVE_PACKET_INITIALIZE_REQ_T;

struct CCLINK_SLAVE_PACKET_INITIALIZE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_INITIALIZE_REQ_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_INITIALIZE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave Task Process Queue
	ulSrc	UINT32	0 ... $2^{32}-1$	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4500	CCLINK_SLAVE_INITIALIZE_REQ – Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information
<b>tData</b>	Structure CCLINK_SLAVE_INITIALIZE_REQ_DATA_T			
	ulReserved	UINT32		Reserved for further use, set to zero

Table 76: CCLINK\_SLAVE\_PACKET\_INITIALIZE\_REQ\_T – Initialization of CC-Link Slave Request

**Packet Structure Reference**

```

typedef struct CCLINK_SLAVE_INITIALIZE_CNF_DATA_Ttag
    CCLINK_SLAVE_INITIALIZE_CNF_DATA_T;

struct CCLINK_SLAVE_INITIALIZE_CNF_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

typedef struct CCLINK_SLAVE_PACKET_INITIALIZE_CNF_Ttag
    CCLINK_SLAVE_PACKET_INITIALIZE_CNF_T;

struct CCLINK_SLAVE_PACKET_INITIALIZE_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_INITIALIZE_CNF_DATA_T tData;
};

```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_INITIALIZE_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4501	CCLINK_SLAVE_INITIALIZE_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	Structure CCLINK_SLAVE_INITIALIZE_CNF_DATA_T			
	ulReserved	UINT32		Reserved for further use, set to zero

Table 77: CCLINK\_SLAVE\_PACKET\_INITIALIZE\_CNF\_T – Initialization of CC-Link Slave Confirmation

## 5.2.2 CCLINK\_SLAVE\_REGISTER\_REQ/CNF – Register Application

This packet is used in order to register to the CC-Link Slave-Task. After this request is performed successfully, indication packets are sent from the CC-Link Slave-Task to the source queue of this request packet.



**Note:** Use this packet preferably when working with linkable object modules. In the context of loadable firmware we recommend to use 'config reload' instead.



**Note:** This packet is used by the AP-task only and will not be routed from the user application to the CC-Link Slave-Task.



**Note:** This packet will no longer be supported by the firmware described in this document after September 1, 2009.

Use the registering functionality described in the netX Dual-Port-Memory Manual instead (RCX\_REGISTER\_APP\_REQ, code 0x2F10).

### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_APP_REGISTER_REQ_DATA_Ttag
    CCLINK_SLAVE_APP_REGISTER_REQ_DATA_T;

struct CCLINK_SLAVE_APP_REGISTER_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

typedef struct CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_Ttag
    CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_T;

struct CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_APP_REGISTER_REQ_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... $2^{32}-1$	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Table 79: CCLINK_SLAVE_REGISTER_REQ – Packet Status/Error
	ulCmd	UINT32	0x4502	CCLINK_SLAVE_REGISTER_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information
<b>tData</b>	structure CCLINK_SLAVE_APP_REGISTER_REQ_DATA_T			
	ulReserved	UINT32		Reserved for further use, set to zero

Table 78: CCLINK\_SLAVE\_PACKET\_APP\_REGISTER\_REQ\_T – Register Application Request

**Packet Status/Error**

Definition / (Value)	Description
TLR_S_OK (0x)	Status ok

Table 79: CCLINK\_SLAVE\_REGISTER\_REQ – Packet Status/Error

**Packet Structure Reference**

```

typedef struct CCLINK_SLAVE_APP_REGISTER_CNF_DATA_Ttag
    CCLINK_SLAVE_APP_REGISTER_CNF_DATA_T;

struct CCLINK_SLAVE_APP_REGISTER_CNF_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

typedef struct CCLINK_SLAVE_PACKET_APP_REGISTER_CNF_Ttag
    CCLINK_SLAVE_PACKET_APP_REGISTER_CNF_T;

struct CCLINK_SLAVE_PACKET_APP_REGISTER_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_APP_REGISTER_CNF_DATA_T tData;
};

```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_APP_REGISTER_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See <i>Table 81: CCLINK_SLAVE_REGISTER_CNF – Packet Status/Error</i>
	ulCmd	UINT32	0x4503	CCLINK_SLAVE_REGISTER_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	Structure CCLINK_SLAVE_APP_REGISTER_CNF_DATA_T			
	ulReserved	UINT32		Reserved for further use, set to zero

Table 80: CCLINK\_SLAVE\_PACKET\_APP\_REGISTER\_CNF\_T – Register Application Confirmation

**Packet Status/Error**

Definition / (Value)	Description
TLR_S_OK (0x)	Status ok
TLR_E_CCLINK_SLAVE_ PACKET_LENGTH (0xC06A0004)	Invalid length in packet.

*Table 81: CCLINK\_SLAVE\_REGISTER\_CNF – Packet Status/Error*

### 5.2.3 CCLINK\_SLAVE\_GET\_BUFFER\_HANDLE\_REQ/CNF – Get Buffer Handle

This packet can be used in order to get the handle for the send and receive triple buffer for data exchange between APS- and CC-Link Slave-Task.



**Note:** Use this packet only when working with linkable object modules. It is not designed for usage in the context of loadable firmware.



**Note:** This packet is used by the AP-task only and will not be routed from the user application to the CC-Link Slave-Task.

The received handles can be used with the macros `TLR_GETEXCHGED_TRIBUFF()` and `TLR_EXCHANGE_TRIBUFF()` in order to exchange data with the CC-Link network.

#### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_Ttag
    CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_T;

struct CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

typedef struct CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_REQ_Ttag
    CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_REQ_T;

struct CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... $2^{32}-1$	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4504	CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information
<b>tData</b>	Structure CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_T			
	ulReserved	UINT32	0	Reserved, set to zero

Table 82: CCLINK\_SLAVE\_PACKET\_GET\_BUFFER\_HANDLE\_REQ\_T – Get Buffer Handle Request

**Packet Structure Reference**

```
typedef struct CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_DATA_Ttag
    CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_DATA_T;

#define CCLINK_SLAVE_SEND_RECV_BUFFER_SIZE 384

struct CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_DATA_Ttag
{
    TLR_UINT32 ulReserved;

    TLR_UINT32 ulRecvBuffer;
    TLR_UINT32 ulSendBuffer;
};

typedef struct CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_CNF_Ttag
    CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_CNF_T;

struct CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	12	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4505	CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	Structure CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_DATA_T			
	ulReserved	UINT32		Reserved for further use
	ulRecvBuffer	UINT32		Receive Buffer
	ulSendBuffer	UINT32		Send Buffer

Table 83: CCLINK\_SLAVE\_GET\_BUFFER\_HANDLE\_CNF\_T – Get Buffer Handle Confirmation

## 5.2.4 CCLINK\_SLAVE\_SET\_BUSPARAM\_REQ/CNF – Set Bus Parameters

This packet is used by the APS-Task in order to set the bus parameters for the CC-Link Slave Task, but can also be sent from the user application. This request will be denied if the configuration lock flag is set (see description of bit D3 in *Table 14: Communication State of Change* of this document).



**Note:** Use this packet preferably when working with linkable object modules. In the context of loadable firmware we recommend to use 'set configuration' or 'warmstart' instead. See section CCLINK\_APS\_SET\_CONFIGURATION\_REQ/CNF – Set Configuration.

### Bus parameter Structure Reference

```
typedef struct CCLINK_SLAVE_BUS_PARAM_Ttag
    CCLINK_SLAVE_BUS_PARAM_T;

#define CCLINK_SLAVE_MIN_SLAVE_STATION_ADDR 0x0001L
#define CCLINK_SLAVE_MAX_SLAVE_STATION_ADDR 0x0040L

#define CCLINK_SLAVE_BAUD_156K           0x0000L
#define CCLINK_SLAVE_BAUD_625K           0x0001L
#define CCLINK_SLAVE_BAUD_2500K          0x0002L
#define CCLINK_SLAVE_BAUD_5M             0x0003L
#define CCLINK_SLAVE_BAUD_10M            0x0004L

#define CCLINK_SLAVE_STATION_TYPE_IO_DEVICE      0x0000L
#define CCLINK_SLAVE_STATION_TYPE_REMOTE_DEVICE 0x0001L

#define CCLINK_SLAVE_NO_STATION_IO_DEVICE        0x0001L

#define CCLINK_SLAVE_MIN_NO_STATIONS_REMOTE_DEVICE 0x0001L
#define CCLINK_SLAVE_MAX_NO_STATIONS_REMOTE_DEVICE 0x0004L

#define CCLINK_SLAVE_CCLINK_VERSION_1  0xL
#define CCLINK_SLAVE_CCLINK_VERSION_2  0x0001L

#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_SINGLE      0x0000L
#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_DOUBLE     0x0001L
#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_QUADRUPLE  0x0002L
#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_OCTUPLE    0x0003L

#define CCLINK_SLAVE_IO_TYPES_POINTS_DEFAULT            0x00000000L
#define CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_DEP_ON_STATION 0x00000001L
#define CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_8POINTS      0x00000002L
#define CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_16POINTS     0x00000003L
#define CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_32POINTS     0x00000004L
#define CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_DEP_ON_STATION 0x00000005L
#define CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_8POINTS      0x00000006L
#define CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_16POINTS     0x00000007L
#define CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_32POINTS     0x00000008L
#define CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_DEP_ON_STATION 0x00000009L
#define CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_8POINTS     0x0000000AL
#define CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_16POINTS    0x0000000BL
#define CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_32POINTS    0x0000000CL
#define CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_DEP_ON_STATION 0x0000000DL
#define CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_8POINTS  0x0000000EL
#define CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_16POINTS 0x0000000FL
#define CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_32POINTS 0x00000010L

struct CCLINK_SLAVE_BUS_PARAM_Ttag
{
    TLR_UINT32    ulSlaveStationAddr;
    TLR_UINT32    ulBaudRate;
}
```

```
TLR_UINT32    ulStationType;
TLR_UINT32    ulNoStation;

TLR_UINT32    ulCcLinkVersion;
TLR_UINT32    ulExtensionCycle;

TLR_UINT32    ulReserved;
TLR_BOOLEAN32 fHoldClrMode;
};

typedef struct CCLINK_SLAVE_ADD_PARAM_Ttag
    CCLINK_SLAVE_ADD_PARAM_T;

#define CCLINK_SLAVE_MAX_VENDOR_CODE    0xFFFFL
#define CCLINK_SLAVE_MAX_MODEL_TYPE    0x00FFL
#define CCLINK_SLAVE_MAX_SW_VERSION    0x003FL

struct CCLINK_SLAVE_ADD_PARAM_Ttag
{
    TLR_UINT32 ulVendorCode;
    TLR_UINT32 ulModelType;
    TLR_UINT32 ulSwVersion;
};
```

Variable	Type	Value / Range	Description
ulSlaveStationAddr	UINT32	1 .. 64	Slave Station Address
ulBaudRate	UINT32	0 1 2 3 4	Baudrate 156 kBaud 625 kBaud 2500 kBaud 5 MBaud 10 MBaud
ulStationType	UINT32	0 1	Station Type Remote I/O Station Remote Device Station
ulNoStation	UINT32	1 1 .. 4	Number of occupied stations Range for Remote I/O Station Range for Remote Device Station
ulCcLinkVersion	UINT32	0 1	Version of CC-Link protocol Version 1 Version 2 (Remote Device Station only)
ulExtensionCycle	UINT32	0  0 1 2 3	Number of extension cycles for CC-Link Version 1 Single setting / one cycle Number of extension cycles for CC-Link Version 2 Single setting / one cycle Double setting / two cycles Quadruple setting / four cycles Octuple setting / eight cycles
ulIoTypesPoints	UINT32	0..16	I/O types and points, see detailed explanation in subsection CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration
fHoldClrMode	BOOLEAN32	0 1	Behavior in case of bus error Clear output data Hold last received output data

Table 84: CCLINK\_SLAVE\_CFG\_BUS\_PARAM\_T - Bus Parameter Configuration

Variable	Type	Range	Description
ulVendorCode	UINT32	0 .. 65535	Vendor Code
ulModelType	UINT32	0 . 255	Model Type
ulSwVersion	UINT32	0 .. 63	Software Version

Table 85: CCLINK\_SLAVE\_CFG\_ADD\_PARAM\_T - Additional Configuration

**Packet Structure Reference**

```

typedef struct CCLINK_SLAVE_SET_BUS_PARAM_REQ_DATA_Ttag
    CCLINK_SLAVE_SET_BUS_PARAM_REQ_DATA_T;

struct CCLINK_SLAVE_SET_BUS_PARAM_REQ_DATA_Ttag
{
    CCLINK_SLAVE_BUS_PARAM_T tBusParam;
    CCLINK_SLAVE_ADD_PARAM_T tAddParam;
};

typedef struct CCLINK_SLAVE_PACKET_SET_BUS_PARAM_REQ_Ttag
    CCLINK_SLAVE_PACKET_SET_BUS_PARAM_REQ_T;

struct CCLINK_SLAVE_PACKET_SET_BUS_PARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_SET_BUS_PARAM_REQ_DATA_T tData;
};

```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_SET_BUSPARAM_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLSLAV E	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -1	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	44	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4506	CCLINK_SLAVE_SET_BUS_PARAM_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information

<b>tData</b>	Structure CCLINK_SLAVE_SET_BUSPARAM_REQ_DATA_T	
	CCLINK_SLAVE_CFG_BUS_PARAM_T	See Table 84: CCLINK_SLAVE_CFG_BUS_PARAM_T - Bus Parameter
	CCLINK_SLAVE_CFG_ADD_PARAM_T	See Table 85: CCLINK_SLAVE_CFG_ADD_PARAM_T - Additional Configuration

Table 86: CCLINK\_SLAVE\_SET\_BUSPARAM\_REQ\_DATA\_T – Set Bus Parameter Request

**Packet Structure Reference**

```

typedef struct CCLINK_SLAVE_SET_BUSPARAM_CNF_DATA_Ttag
    CCLINK_SLAVE_SET_BUSPARAM_CNF_DATA_T;

struct CCLINK_SLAVE_SET_BUSPARAM_CNF_DATA_Ttag
{
    CCLINK_SLAVE_CFG_BUS_PARAM_T tCfgBusParam;
    CCLINK_SLAVE_CFG_ADD_PARAM_T tCfgAddParam;
};

typedef struct CCLINK_SLAVE_PACKET_SET_BUSPARAM_CNF_Ttag
    CCLINK_SLAVE_PACKET_SET_BUSPARAM_REQ_T;

struct CCLINK_SLAVE_PACKET_SET_BUSPARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_SET_BUSPARAM_REQ_DATA_T tData;
};

```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_SET_BUSPARAM_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	44	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4507	CCLINK_SLAVE_SET_BUSPARAM_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	Structure CCLINK_SLAVE_SET_BUSPARAM_REQ_DATA_T			
	CCLINK_SLAVE_CFG_BUS_PARAM_T	See Table 84: CCLINK_SLAVE_CFG_BUS_PARAM_T - Bus Parameter		
	CCLINK_SLAVE_CFG_ADD_PARAM_T	See Table 85: CCLINK_SLAVE_CFG_ADD_PARAM_T - Additional Configuration		

Table 87: CCLINK\_SLAVE\_PACKET\_SET\_BUSPARAM\_CNF\_T –Set Bus Parameter Confirmation

## 5.2.5 CCLINK\_SLAVE\_STARTSTOP\_REQ/CNF – Start/Stop Communication with Network

This packet starts or stops the communication with the CC-Link network depending on the value of the `ulMode` parameter.



**Note:** This packet is obsolete and will not longer supported after September,1, 2009. It is replaced by packet `RCX_START_STOP_COMM_REQ/CNF` which contains identical functionality. Do not use this packet for all new developments.

### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_STARTSTOP_REQ_DATA_Ttag
    CCLINK_SLAVE_STARTSTOP_REQ_DATA_T;

#define CCLINK_SLAVE_STOP_CCLINK      0xL
#define CCLINK_SLAVE_START_CCLINK    0x0001L

struct CCLINK_SLAVE_STARTSTOP_REQ_DATA_Ttag
{
    TLR_UINT32 ulMode;
};

typedef struct CCLINK_SLAVE_PACKET_STARTSTOP_REQ_Ttag
    CCLINK_SLAVE_PACKET_STARTSTOP_REQ_T;

struct CCLINK_SLAVE_PACKET_STARTSTOP_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_STARTSTOP_REQ_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_STARTSTOP_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -1	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4508	CCLINK_SLAVE_STARTSTOP_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information
<b>tData</b>	Structure CCLINK_SLAVE_STARTSTOP_REQ_DATA_T			
	ulMode	UINT32	0 1	Depending on this assignment, communication is either started or stopped: 0 Stop communication with Network 1 Start communication with Network

Table 88: CCLINK\_SLAVE\_PACKET\_STARTSTOP\_REQ\_T – Start/Stop Communication Request

**Packet Structure Reference**

```
typedef struct CCLINK_SLAVE_STARTSTOP_CNF_DATA_Ttag
    CCLINK_SLAVE_STARTSTOP_CNF_DATA_T;

#define CCLINK_SLAVE_STOP_CCLINK      0xL
#define CCLINK_SLAVE_START_CCLINK     0x0001L

struct CCLINK_SLAVE_STARTSTOP_CNF_DATA_Ttag
{
    TLR_UINT32 ulMode;
};

typedef struct CCLINK_SLAVE_PACKET_STARTSTOP_CNF_Ttag
    CCLINK_SLAVE_PACKET_STARTSTOP_CNF_T;

struct CCLINK_SLAVE_PACKET_STARTSTOP_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_STARTSTOP_CNF_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_STARTSTOP_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x4509	CCLINK_SLAVE_STARTSTOP_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	structure CCLINK_SLAVE_STARTSTOP_CNF_DATA_T			
	ulMode	UINT32		Depending on this assignment, communication is either started or stopped:
			0	Stop communication with network
			1	Start communication with network

Table 89: CCLINK\_SLAVE\_PACKET\_STARTSTOP\_CNF\_T – Start/Stop Communication Confirmation

## 5.2.6 CCLINK\_SLAVE\_GET\_CCL\_STATUS\_REQ/CNF – Get CC-Link Status

This request can be used in order to get the status of the CC-Link Master and Slave.

### **Packet Structure Reference**

```
typedef struct CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_Ttag
    CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_T;

struct CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

typedef struct CCLINK_SLAVE_PACKET_GET_CCL_STATUS_REQ_Ttag
    CCLINK_SLAVE_PACKET_GET_CCL_STATUS_REQ_T;

struct CCLINK_SLAVE_PACKET_GET_CCL_STATUS_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_GET_CCL_STATUS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -1	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x450A	CCLINK_SLAVE_GET_CCL_STATUS_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information
<b>tData</b>	Structure CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_T			
	ulReserved	UINT32		Reserved for further use, set to zero

Table 90: CCLINK\_SLAVE\_PACKET\_GET\_CCL\_STATUS\_REQ\_T – Get CC-Link Status Request

**Packet Structure Reference**

```

typedef struct CCLINK_SLAVE_GET_CCL_STATUS_CNF_DATA_Ttag
    CCLINK_SLAVE_GET_CCL_STATUS_CNF_DATA_T;

#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_MSK          0x0001L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK 0x0002L
#define CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK              0x0004L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK        0x0008L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK  0x0010L
#define CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK             0x0060L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK          0x0080L
#
define CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK    0x0F00L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK  0xF000L

#define CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK                   0x0001L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVALID_NUM_OF_POINTS_MSK 0x0002L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK           0x0004L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK        0x0008L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK     0x0010L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK        0x0020L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK                         0x0040L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK                    0x0080L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK        0x0100L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK  0x0200L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK               0x0400L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK                         0x0800L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK    0x1000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK             0x2000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK       0xC000L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SINGLE_MSK        0xL
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_DOUBLE_MSK        0x4000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_QUADRUPLE_MSK     0x8000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_OCTUPLE_MSK       0xC000L

struct CCLINK_SLAVE_GET_CCL_STATUS_CNF_DATA_Ttag
{
    TLR_UINT32 ulReserved;
    TLR_UINT32 ulMasterState;
    TLR_UINT32 ulSlaveState;
};

typedef struct CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_Ttag
    CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_T;

struct CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_GET_CCL_STATUS_CNF_DATA_T tData;
};

```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	12	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x450B	CCLINK_SLAVE_GET_CCL_STATUS_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	structure CCLINK_SLAVE_GET_CCL_STATUS_CNF_DATA_T			
	ulReserved	UINT32		Reserved for further use
	ulMasterState	UINT32		See section <i>Extended Status</i>
	ulSlaveState	UINT32		See section <i>Extended Status</i>

Table 91: CCLINK\_SLAVE\_PACKET\_GET\_CCL\_STATUS\_CNF\_T – Get CC-Link Status Confirmation

## 5.2.7 CCLINK\_SLAVE\_CHANGE\_SLAVE\_STATUS\_REQ/CNF – Change CC-Link Slave Status

This request can be used in order to change several flags within the status of the CC-Link Slave.



**Note:** The switch change flag cannot be changed from the user application if the rotary switches are handled by the APS-Task.

### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ_DATA_Ttag
    CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_T;

#define CCLINK_SLAVE_EVAL_FUSE_STATE_PARAM_MSK      0x0001L
#define CCLINK_SLAVE_EVAL_SWITCH_CHANGE_PARAM_MSK  0x0002L
#define CCLINK_SLAVE_EVAL_WATCHDOG_ERROR_PARAM_MSK 0x0004L

struct CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ_DATA_Ttag
{
    TLR_UINT32      ulFlags;

    TLR_BOOLEAN32  fFuseStatus;
    TLR_BOOLEAN32  fSwitchChange;
    TLR_BOOLEAN32  fWatchdogError;
};

typedef struct CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_REQ_Ttag
    CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_REQ_T;

struct CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... $2^{32}-1$	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	16	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		<i>See section 6.2 Status/Error codes CC-Link Slave-Task</i>
	ulCmd	UINT32	0x450C	CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information

tData	Structure CCLINK_SLAVE_CHANEGE_SLAVE_STATUS_REQ_DATA_T			
	ulFlags	UINT32		Flags  Bit 0:Parameter <code>fFuseStatus</code> will be evaluated if this bit is set  Bit 1:Parameter <code>fSwitchChange</code> will be evaluated if this bit is set  Bit 2:Parameter <code>fWatchdogError</code> will be evaluated if this bit is set  Bit 3 .. 31: Reserved for further use, set to zero
	fFuseStatus	UINT32	FALSE TRUE	This parameter will be evaluated if corresponding bit in parameter <code>ulFlags</code> is set  Fuse status set to ok Fuse status set to error
	fSwitchChange	UINT32	FALSE TRUE	This parameter will be evaluated if corresponding bit in parameter <code>ulFlags</code> is set  Switch change status set to no change Switch change status set to change detected
	fWatchdogError	UINT32	FALSE TRUE	This parameter will be evaluated if corresponding bit in parameter <code>ulFlags</code> is set  Clear watchdog error Set watchdog error

Table 92: CCLINK\_SLAVE\_PACKET\_CHANGE\_SLAVE\_STATUS\_REQ\_T – Change CC-Link Slave Status Request

**Packet Structure Reference**

```
typedef struct CCLINK_SLAVE_CHANGE_SLAVE_STATUS_CNF_DATA_Ttag
    CCLINK_SLAVE_CHANGE_SLAVE_STATUS_CNF_DATA_T;

struct CCLINK_SLAVE_CHANGE_SLAVE_STATUS_CNF_DATA_Ttag
{
    TLR_UINT32 ulFlags;

    TLR_BOOLEAN32 fFuseStatus;
    TLR_BOOLEAN32 fSwitchChange;
    TLR_BOOLEAN32 fWatchdogError;
};

typedef struct CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_Ttag
    CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_T;

struct CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead;
    CCLINK_SLAVE_CHANGE_SLAVE_STATUS_CNF_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	16	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x450D	CCLINK_SLAVE_CHANGE_SLAVE_STATUS_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change

tData	structure CCLINK_SLAVE_CHNAGE_SLAVE_STATUS_CNF_DATA_T			
	ulFlags	UINT32		Flags  Bit 0:Parameter fFuseStatus will be evaluates if this bit is set  Bit 1:Parameter fSwitchChange will be evaluates if this bit is set  Bit 2:Parameter fWatchdogError will be evaluates if this bit is set  Bit 3 .. 31: Reserved for further use, set to zero
	fFuseStatus	UINT32	FALSE TRUE	This parameter will be evaluated if bit 0 in parameter ulFlags is set  Fuse status set to ok Fuse status set to error
	fSwitchChange	UINT32	FALSE TRUE	This parameter will be evaluated if bit 1 in parameter ulFlags is set  Switch change status set to no change Switch change status set to change detected
	fWatchdogError	UINT32	FALSE TRUE	This parameter will be evaluated if bit 2 in parameter ulFlags is set  Clear watchdog error Set watchdog error

**Table 93:** CCLINK\_SLAVE\_PACKET\_CHANGE\_SLAVE\_STATUS\_CNF\_T – Change CC-Link Slave Status Confirmation

## 5.2.8 CCLINK\_SLAVE\_GET\_BUS\_PARAM\_REQ/CNF – Get Bus Parameters

This request can be used in order to get the current bus parameters.

### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_PACKET_GET_BUS_PARAM_REQ_Ttag
  CCLINK_SLAVE_PACKET_GET_BUS_PARAM_REQ_T;

struct CCLINK_SLAVE_PACKET_GET_BUS_PARAM_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
};
```

### Packet Description

Structure Information CCLINK_SLAVE_PACKET_GET_BUS_PARAM_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_CCLSLAV E	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -1	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x450E	CCLINK_SLAVE_GET_BUS_PARAM_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information

Table 94: CCLINK\_SLAVE\_PACKET\_GET\_BUS\_PARAM\_REQ\_T – Get Bus Parameter Request

**Packet Structure Reference**

```
typedef struct CCLINK_SLAVE_GET_BUS_PARAM_CNF_DATA_Ttag
    CCLINK_SLAVE_GET_BUS_PARAM_CNF_DATA_T;

struct CCLINK_SLAVE_GET_BUS_PARAM_CNF_DATA_Ttag
{
    CCLINK_SLAVE_BUS_PARAM_T tBusParam;
    CCLINK_SLAVE_ADD_PARAM_T tAddParam;
};

typedef struct CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_Ttag
    CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_T;

struct CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;
    CCLINK_SLAVE_GET_BUS_PARAM_CNF_DATA_T tData;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	44	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x450F	CCLINK_SLAVE_GET_BUS_PARAM_CNF - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change
<b>tData</b>	structure CCLINK_SLAVE_GET_BUS_PARAM_CNF_DATA_T			
	CCLINK_SLAVE_CFG_BUS_PARAM_T	See Table 84: CCLINK_SLAVE_CFG_BUS_PARAM_T - Bus Parameter		
	CCLINK_SLAVE_CFG_ADD_PARAM_T	See Table 85: CCLINK_SLAVE_CFG_ADD_PARAM_T - Additional Configuration		

Table 95: CCLINK\_SLAVE\_PACKET\_GET\_BUS\_PARAM\_CNF\_T – Get Bus Parameter Confirmation

## 5.2.9 CCLINK\_SLAVE\_STATE\_CHANGE\_IND/RES – Change of State Indication

This indication packet signifies a change of the state of the CC-Link Slave-Task or the CC-Link network. The indication delivers two important blocks containing status information about the CC-Link Slave, namely

- The slave state
- The extended slave state

These blocks delivering information about the change of state are described in detail below.



**Note:** Use this packet only when working with linkable object modules. It is not designed for usage in the context of loadable firmware.



**Note:** This indication is used by the AP-Task in order to set status information in the DPM and will not be routed to the user application.

In order to be able to receive this indication, the CCLINK\_SLAVE\_REGISTER\_REQ/CNF – Register Application request described in section 5.2.2 of this document has to be executed by the AP-Task.

### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_STATE_CHANGE_IND_DATA_Ttag
  CCLINK_SLAVE_STATE_CHANGE_IND_DATA_T;

__PACKED_PRE struct CCLINK_SLAVE_STATE_CHANGE_IND_DATA_Ttag
{
  CCLINK_SLAVE_SLAVE_STATE_T      tSlaveState;
  CCLINK_SLAVE_EXTENDED_STATE_T   tExtendedState;
}__PACKED_POST;

typedef struct CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_Ttag
  CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_T;

__PACKED_PRE struct CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_Ttag
{
  TLR_PACKET_HEADER_T              tHead;
  CCLINK_SLAVE_STATE_CHANGE_IND_DATA_T tData;
}__PACKED_POST;
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_T				
Type: Indication				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle of AP-Task Process Queue
	ulSrc	UINT32		Source Queue-Handle of CC-Link Slave-Task Process Queue
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	52	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x451E	CCLINK_SLAVE_STATE_CHANGE_IND - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information
<b>tData</b>	Structure CCLINK_SLAVE_STATE_CHANGE_IND_DATA_T			
	tSlaveState	CCLINK_SLAVE_SLAVE_STATE_T		Structure for slave state, see explanation below.
	tExtended State	CCLINK_SLAVE_EXTENDED_STATE_T		Structure for extended slave state, see explanation below.

Table 96: CCLINK\_SLAVE\_PACKET\_STATE\_CHANGE\_IND\_T – Change of State Indication

**CC-Link Slave State Structure Reference**

```
typedef struct CCLINK_SLAVE_SLAVE_STATE_Ttag
    CCLINK_SLAVE_SLAVE_STATE_T;

#define CCLINK_SLAVE_STATE_FLAG_RDY          0x0001L
#define CCLINK_SLAVE_STATE_FLAG_RUN         0x0002L
#define CCLINK_SLAVE_STATE_FLAG_COM         0x0004L
#define CCLINK_SLAVE_STATE_FLAG_BUS_ON      0x0008L
#define CCLINK_SLAVE_STATE_FLAG_COMM_ERROR 0x0010L

struct CCLINK_SLAVE_SLAVE_STATE_Ttag
{
    TLR_UINT32    ulCcLinkState;

    TLR_UINT32    ulFlags;
    TLR_UINT32    ulErrorCount;

    TLR_UINT32    ulCommError;
    TLR_UINT32    ulStaLedState;
    TLR_UINT32    ulErrLedState;

    TLR_UINT32    ulIoByteCnt;

    TLR_UINT32    aulReserved[3];
};
```

```

typedef struct CCLINK_SLAVE_EXTENDED_STATE_Ttag
    CCLINK_SLAVE_EXTENDED_STATE_T;

#define CCLINK_SLAVE_EXT_STATE_FLAG_WDG 0x0001L
#define CCLINK_SLAVE_EXT_STATE_CTRL    0x0002L
#define CCLINK_SLAVE_EXT_STATE_NRDY    0x0004L

#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_MSK      0x0001L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK 0x0002L
#define CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK           0x0004L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK     0x0008L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK 0x0010L
#define CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK         0x0060L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK      0x0080L

#define CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK 0x0F00L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK 0xF000L

#define CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK                0x0001L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVALID_NUM_OF_POINTS_MSK 0x0002L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK       0x0004L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK     0x0008L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK  0x0010L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK     0x0020L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK                     0x0040L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK                0x0080L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK    0x0100L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK 0x0200L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK           0x0400L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK                     0x0800L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK 0x1000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK        0x2000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK   0xC000L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SINGLE_MSK    0xL
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_DOUBLE_MSK    0x4000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_QUADRUPLE_MSK 0x8000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_OCTUPLE_MSK   0xC000L

__PACKED_PRE struct CCLINK_SLAVE_EXTENDED_STATE_Ttag
{
    TLR_UINT32 ulFlags;

    TLR_UINT32 ulMasterState;
    TLR_UINT32 ulSlaveState;

    TLR_UINT32 ulRxByteCount;
    TLR_UINT32 ulRWrByteCount;

    TLR_UINT32 ulRyByteCount;
    TLR_UINT32 ulRWwByteCount;
}__PACKED_POST;

```

The extended slave state is described in detail in [section 3.3.2 “Extended Status”](#).

**Packet Structure Reference**

```

typedef struct CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_Ttag
    CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_T;

struct CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_Ttag
{
    TLR_PACKET_HEADER_T tHead;
};

```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_T				
Type: Response				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32		Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x451F	CCLINK_SLAVE_STATE_CHANGE_RES - Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change

Table 97: CCLINK\_SLAVE\_PACKET\_STATE\_CHANGE\_RES\_T – Change of State Response

## 5.2.10 CCLINK\_SLAVE\_SET\_WATCHDOG\_FAIL\_REQ/CNF – Set Watchdog Fail

This packet is used by the AP-Task in order to inform the CC-Link Slave-Task that a watchdog failure has been detected. The CC-Link Slave-Task stops the communication with the CC-Link network. After a watchdog error has been set, the slave has to be reinitialized before further communication is possible.



**Note:** Use this packet only when working with linkable object modules. It is not designed for usage in the context of loadable firmware.



**Note:** This packet is used by the AP-task only and will not be routed from the user application to the CC-Link Slave-Task.

### Packet Structure Reference

```
typedef struct CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_REQ_Ttag
    CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_REQ_T;

struct CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;
};
```

### Packet Description

Structure Information CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_CCLSLAVE	Destination Queue-Handle of CC-Link Slave-Task Process Queue
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -1	Source Queue-Handle of AP-Task Process Queue
	ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
	ulSrcId	UINT32	0 ... 2 <sup>32</sup> -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 6.2 Status/Error codes CC-Link Slave-Task
	ulCmd	UINT32	0x45AA	CCLINK_SLAVE_SET_WATCHDOG_FAIL_REQ - Command
	ulExt	UINT32	0	Reserved
	ulRout	UINT32	x	Routing Information

Table 98: CCLINK\_SLAVE\_PACKET\_SET\_WATCHDOG\_FAIL\_REQ\_T – Set Watchdog Fail Request

**Packet Structure Reference**

```
typedef struct CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_Ttag
    CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_T;

struct CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;
};
```

**Packet Description**

Structure Information CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
<b>tHead</b>	Structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle, untouched
	ulSrc	UINT32		Source Queue-Handle, untouched
	ulDestId	UINT32		Destination End Point Identifier, untouched
	ulSrcId	UINT32		Source End Point Identifier, untouched
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		<i>See section 6.2 Status/Error codes CC-Link Slave-Task</i>
	ulCmd	UINT32	0x45AB	CCLINK_SLAVE_SET_WATCHDOG_FAIL_CNF-Command
	ulExt	UINT32		Extension, reserved
	ulRout	UINT32		Routing Information, do not change

Table 99: CCLINK\_SLAVE\_PACKET\_SET\_WATCHDOG\_FAIL\_CNF\_T – Set Watchdog Fail Confirmation

## 6 Status/Error Codes Overview

### 6.1 Status/Error Codes CC-Link APS-Task

Hexadecimal Value	Definition Description
0x	TLR_S_OK Status ok
0x406B0002	TLR_I_CCLINK_APS_OPEN_DBM_FILE Failed to open configuration database.
0xC06B0003	TLR_E_CCLINK_APS_DATASET Failed to open configuration dataset.
0xC06B0004	TLR_E_CCLINK_APS_TABLE_GLOBAL Failed to open GLOBAL configuration dataset.
0xC06B0005L	TLR_E_CCLINK_APS_TABLE_CCLS_INI Failed to open CCLS_INI configuration dataset.
0xC06B0006	TLR_E_CCLINK_APS_WATCHDOG_PARAMETER Invalid parameter for watchdog supervision.
0xC06B0007	TLR_E_CCLINK_APS_SIZE_TABLE_CCLS_INI Invalid size of CCLS_INI configuration dataset.
0xC06B000A	TLR_E_CCLINK_APS_DATABASE_FOUND Configuration database found.
0xC06B000BL	TLR_E_CCLINK_APS_SLAVE_STATION_ADDR_PARAMETER Invalid parameter for slave station address.
0xC06B000CL	TLR_E_CCLINK_APS_BAUDRATE_PARAMETER Invalid parameter for baudrate.
0xC06B000D	TLR_E_CCLINK_APS_NO_STATION_PARAMETER Invalid parameter for number of stations.
0xC06B000E	TLR_E_CCLINK_APS_MODE_PARAMETER Invalid parameter for mode.
0xC06B000F	TLR_E_CCLINK_APS_VENDOR_CODE_PARAMETER Invalid parameter for vendor code.
0xC06B0010	TLR_E_CCLINK_APS_MODEL_CODE_PARAMETER Invalid parameter for model code.
0xC06B0011	TLR_E_CCLINK_APS_SW_VERSION_PARAMETER Invalid parameter for software version.
0xC06B0012	TLR_E_CCLINK_APS_MODEL_TYPE_PARAMETER Invalid parameter for model type.
0xC06B0013	TLR_E_CCLINK_APS_IO_MODE_PARAMETER Invalid parameter for IO mode.
0xC06B0015	TLR_E_CCLINK_APS_INVALID_STATE Request not allowed in current state.
0xC06B0016	TLR_E_CCLINK_APS_PARAM_CYCLETIME Invalid parameter for cycle time.
0xC06B0017	TLR_E_CCLINK_APS_PARAM_CHN_INSTANCE Invalid parameter for DPM instance.

0xC06B0018	TLR_E_CCLINK_APS_SET_SWITCH_CHANGE_NOT_ALLOWED Change switch state not allowed.
0xC06B0019	TLR_E_CCLINK_APS_CCLINK_VERSION_PARAMETER Invalid parameter for CC-Link version.
0xC06B001A	TLR_E_CCLINK_APS_STATION_TYPE_PARAMETER Invalid parameter for station type.
0xC06B001B	TLR_E_CCLINK_APS_STATION_ADDR_WITH_NO_STATIONS_PARAMETER Invalid parameter for station address in combination with number of occupied stations.
0xC06B001C	TLR_E_CCLINK_APS_EXTENSION_CYCLE_PARAMETER Invalid parameter extension cycle.
0xC06B001D	TLR_E_CCLINK_APS_STATION_TYPE_WITH_CCLINK_VERSION_PARAMETER Invalid parameter for station type in combination with CC-Link version.
0xC06B001E	TLR_E_CCLINK_APS_PARAM_QUEUE_ELEMENT Invalid parameter for number of queue elements.
0xC06B001F	TLR_E_CCLINK_APS_PARAM_POOL_ELEMENT Invalid parameter for number of pool elements.
0xC06B0020	TLR_E_CCLINK_APS_PARAM_SWITCH Invalid parameter for switch parameter.
0xC06B0021	TLR_E_CCLINK_APS_PARAM_IO_TYPES_POINTS Invalid parameter for number of I/O types and I/O points.

Table 100: Status/Error Codes CC-Link APS-Task

## 6.2 Status/Error codes CC-Link Slave-Task

Hexadecimal Value	Definition Description
0x	TLR_S_OK Status ok
0xC06A0003	TLR_I_CCLINK_SLAVE_ALREADY_IN_STATE Slave is already in requested state.
0xC06A0005	TLR_E_CCLINK_SLAVE_DATA_COUNT Invalid data count.
0xC06A0006	TLR_E_CCLINK_SLAVE_DATA_OFFSET Invalid data offset.
0xC06A0007	TLR_E_CCLINK_SLAVE_INIT_BUFFER Initialization of buffer failed.
0xC06A0008	TLR_E_CCLINK_SLAVE_INVALID_STATE Command is not allowed in current state.
0xC06A0009	TLR_E_CCLINK_SLAVE_MODE Invalid mode in command.
0xC06A000A	TLR_E_CCLINK_SLAVE_PARAM_BAUDRATE Invalid Baudrate.
0xC06A000B	TLR_E_CCLINK_SLAVE_PARAM_STATION_ADDR Invalid station address for CC-Link Slave.
0xC06A000C	TLR_E_CCLINK_SLAVE_PARAM_NO_STATIONS Invalid parameter for number of stations.
0xC06A000D	TLR_E_CCLINK_SLAVE_PARAM_VENDOR_CODE Invalid parameter for vendor code.
0xC06A000E	TLR_E_CCLINK_SLAVE_PARAM_MODEL_CODE Invalid parameter for model code.
0xC06A000F	TLR_E_CCLINK_SLAVE_PARAM_SW_VERSION Invalid parameter for software version.
0xC06A0010	TLR_E_CCLINK_SLAVE_PARAM_MODEL_TYPE Invalid parameter for model type.
0xC06A0011	TLR_E_CCLINK_SLAVE_PARAM_STATION_TYPE Invalid parameter for station type.
0xC06A0012	TLR_E_CCLINK_SLAVE_PARAM_CYCLETIME Invalid parameter for cycle time.
0xC06A0013	TLR_E_CCLINK_SLAVE_PARAM_XC_INSTANCE Invalid parameter for XC-Instance.
0xC06A0014	TLR_E_CCLINK_SLAVE_PARAM_STATION_ADDR_WITH_NO_STATIONS Invalid parameter for station address in combination with number of occupied stations.
0xC06A0015	TLR_E_CCLINK_SLAVE_PARAM_CCLINK_VERSION Invalid parameter for CC-Link version.

0xC06A0016	TLR_E_CCLINK_SLAVE_PARAM_EXTENSION_CYCLE Invalid parameter for extension cycle.
0xC06A0017	TLR_E_CCLINK_SLAVE_PARAM_STATION_TYPE_WITH_CCLINK_VERSION Invalid parameter for station type in combination with CC-Link version.
0xC06A0018	TLR_E_CCLINK_SLAVE_PARAM_QUEUE_ELEMENT Invalid parameter for number of queue elements.
0xC06A0019	TLR_E_CCLINK_SLAVE_PARAM_POOL_ELEMENT Invalid parameter for number of pool elements.
0xC06A001A	TLR_E_CCLINK_SLAVE_PARAM_IO_TYPES_POINTS Invalid parameter for number of I/O types and I/O points.

Table 101: Status/Error Codes CC-Link Slave-Task

## 7 Contact

### 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](mailto:info@hilscher.com)

#### Support

Phone: +49 (0) 6190 9907-99  
E-Mail: [de.support@hilscher.com](mailto: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](mailto:info@hilscher.cn)

#### Support

Phone: +86 (0) 21-6355-5161  
E-Mail: [cn.support@hilscher.com](mailto: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](mailto:info@hilscher.fr)

#### Support

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

#### India

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

#### Italy

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

#### Support

Phone: +39/02 25007068  
E-Mail: [it.support@hilscher.com](mailto:it.support@hilscher.com)

#### Japan

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

#### Support

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

#### Korea

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

#### Switzerland

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

#### Support

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

#### USA

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

#### Support

Phone: +1 630-505-5301  
E-Mail: [us.support@hilscher.com](mailto:us.support@hilscher.com)