Protocol API

CC-Link Slave

V2.12.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC071101API11EN | Revision 11 | English | 2017-01 | Released | Public

Table of Contents

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 List of Revisions

Rev	Date	Name	Revisions
9	2014-09-01	RG	Firmware/stack version V2.11.x Reference to netX Dual-Port Memory Interface Manual Revision 12. Error corrections in description of hardware switches Removed description of rcX packages
10	2015-01-05	RG/ES	Firmware/stack version V2.11.x Reference to netX Dual-Port Memory Interface Manual Revision 12. New packet `CCLINK_APS_SET_DATA_LOOP_REQ/CNF` added for resending input data from master unchanged.
11	2017-01-16	HH	Firmware/stack version V2.12.0 CC-Link Slave V2.12.0 has the same API as V2.11.x. Section Extended Status contains the CC-Link specific status.

Table 1: List of Revisions

1.3 System Requirements

This software package has the following environmental system requirements:

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

1.4 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

1.5 Specifications

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

1.5.1 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 2: Names of Tasks in CC-Link Slave Firmware

1.5.2 Technical Data

The data below applies to the CC-Link firmware and stack version V2.11.x.x. The firmware/stack supports 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 51	yes (supported since stack V2.9.1.x)
netX 52	yes (supported since stack V2.10.1.x)
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

1.6 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 3: 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.

1.7 References

This document refers to the following documents:

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

[2] CC-Link Partner Association: CC-Link Specification BAP-05026-J, V2.00

[3] Hilscher Gesellschaft für Systemautomation mbH: Operating Instruction Manual, Tag List Editor, Viewing and Editing Tags in NXF/NXO/BSL Files, V1.2, Revision 5, 2015.

Table 4: References

1.8 Legal Notes

1.8.1 Copyright

1.8.2 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.

1.8.3 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.

1.8.4 Export

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different counters, 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 Getting started / Configuration

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

2.1 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 Name	Page
Configure CC-Link Slave	CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration	30
Start/Stop network communication	CCLINK_SLAVE_STARTSTOP_REQ/CNF – Start/Stop Communication with Network	53
Initialization	CCLINK_SLAVE_INITIALIZE_REQ/CNF – Initialization of CC-Link Slave	40

Table 5: Overview about Essential Functionality

2.2 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	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: Reserved for further use, set to zero BIT 4: ADDRESS_SWITCH Should be set when hardware address switch is used and there is no TAG present. BIT 5: BAUD_SWITCH Should be set when hardware baudrate switch is used and there is no TAG present. BIT 6 - 31: Reserved for further use, set to zero
CcLink Flags	CC-Link flags	Bit 0: Vendor Code DISABLED/ ENABLED Parameter ulVendorCode will be evaluated if this bit is set. Otherwise the default value will be used Bit 1: Model Type DISABLED/ ENABLED Parameter ulModelType will be evaluated if this bit is set. Otherwise the default value will be used Bit 2: SW Version DISABLED/ ENABLED Parameter ulSwVersion will be evaluated if this bit is set. Otherwise the default value will be used BIT 3 - 31: Reserved for further use, set to zero
Watchdog Time	Watchdog supervision time [ms] within which the device watchdog must be retriggered from the application program while the application program monitoring is activated. Watchdog supervision deactivated Watchdog supervision activated	 0 20 .. 65535
Slave Station Address	Station address of CC-Link Slave Warmstart parameter Parameter taken from rotary switch (Allowed for devices with rotary switches) **Note:** The number of occupied stations and station address must not exceed the maximum station address (Address + Number of occupied stations - 1 <= 64).	 1 .. 64 255
Baud rate	Network transmission rate **Note:** Parameter is taken from rotary switch if station address is set to 255 (Allowed for devices with rotary switches).	See *Table 7: Available Baud Rate Values*

Parameter	Meaning	Range of Value / Value
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: **Note:** 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 `ulCcLinkFlags` parameter is set)	0 . 65535
Model Type	Model type (If corresponding bit in parameter `ulCcLinkFlags` parameter is set)	0 .. 255
Sw Version	Software version (If corresponding bit in parameter `ulCcLinkFlags` parameter is set)	0 .. 63

Table 6: 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 7: Available Baud Rate Values

2.2.1 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 packet's name is CCLINK_APS_SET_CONFIGURATION_REQ for the request packet and CCLINK_APS_SET_CONFIGURATION_CNF for the confirmation packet.
- 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:

Contrary to the situation when receiving a set configuration command, on every received warmstart packet an automatic channel-init is performed.

The entire parameter set is completely delivered within the CCLINK_APS_SET_CONFIGURATION_CNF or CCLINK_APS_WARMSTART_CNF packet.

2.3 Input and Output Data

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

2.3.1 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 8: 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 9: 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 10: 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 11: 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 12: Input and Output Data for Remote Device Station with Four Occupied Stations

2.3.2 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 13: 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 14: 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 15: 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 16: 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 17: 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 18: 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 19: 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 20: 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 21: 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 22: 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 23: 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 24: 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 25: 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 26: 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 27: 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 28: Input and Output Data for Remote Device Station with Four Occupied Stations, Octuple Setting

2.4 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:

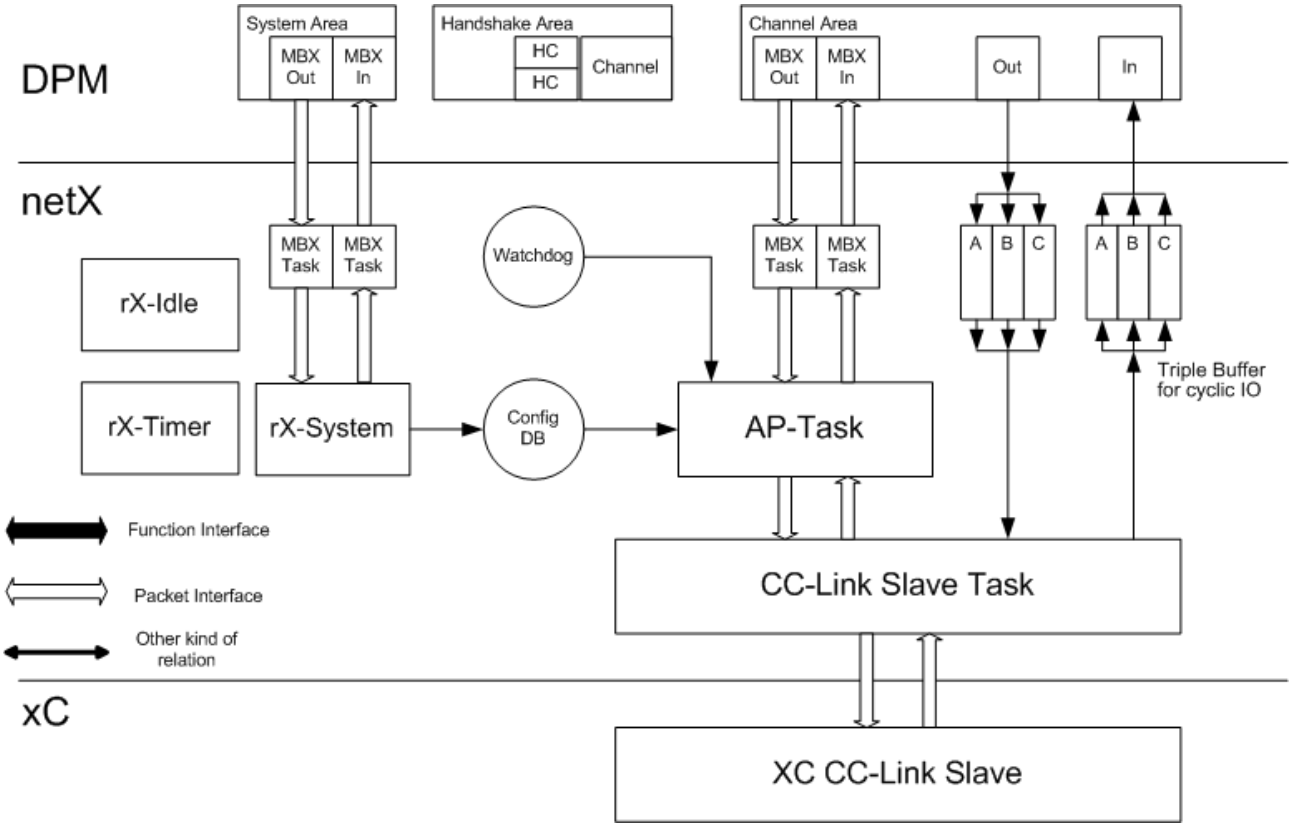

Figure 1: Internal Structure of CC-Link Slave Firmware

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.

3 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.

3.1 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-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 29: 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_SENDPACKET_FIFO/LIFO()` for sending a packet to the CC-Link APS-Task.

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

Packet	Command code (REQ/CNF or IND/RES)	Page
`CCLINK_APS_WARMSTART_REQ/CNF` – Set Warmstart Parameters	`0x4600/0x4601`	23
`CCLINK_APS_SET_CONFIGURATION_REQ/CNF` – Set Configuration	`0x4604/0x4605`	30
`CCLINK_APS_SET_DATA_LOOP_REQ/CNF` – Set Data Loop	`0x46E0/0x46E1`	36

Table 30: Overview over the Packets of the APS-Task of the CC-Link Slave Protocol Stack

3.1.1 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.

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   0x00000001L
#define CCLINK_APS_SYS_FLAG_IO_STATUS_ENABLED        0x00000002L
#define CCLINK_APS_SYS_FLAG_IO_STATUS_32_BIT         0x00000004L

#define CCLINK_APS_SYS_FLAG_ADDRESS_SWITCH           0x00000010L /* Switch for address */
#define CCLINK_APS_SYS_FLAG_BAUD_SWITCH              0x00000020L /* Switch for baud    */

#define CCLINK_APS_WD_OFF                            0x00000000L
#define CCLINK_APS_WD_MIN_TIMEOUT                    0x00000014L
#define CCLINK_APS_WD_MAX_TIMEOUT                    0x0000FFFFL

#define CCLINK_APS_CCLS_FLAGS_CFG_VENDOR_CODE        0x00000001L
#define CCLINK_APS_CCLS_FLAGS_CFG_MODEL_TYPE         0x00000002L
#define CCLINK_APS_CCLS_FLAGS_CFG_SW_VERSION         0x00000004L

#define CCLINK_APS_STATION_ADDR_CFG_ROTARY_SWITCH    0x000000FFL

struct CCLINK_APS_WARMSTART_REQ_DATA_Ttag
{
  TLR_UINT32    ulSystemFlags;      /* System flags   */
  TLR_UINT32    ulCcLinkFlags;      /* CC-Link Flags  */
  TLR_UINT32    ulWdgTime;          /* Watchdog time  */
  TLR_UINT32    ulSlaveStationAddr; /* Node ID        */
  TLR_UINT32    ulBaudRate;         /* BaudRate       */
  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
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error Codes CC-Link APS-Task on page 86.
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
structure CCLINK_APS_WARMSTART_REQ_DATA_T			
ulSystemFlags	UINT32		System Flags BIT 0: AUTOSTART / APPLICATION CONTROLLED
		0	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.
		1	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
ulCcLinkFlags	UINT32		CC-Link Flags
		0 1	Bit 0: CONFIG VENDOR CODE DISABLED/ENABLED Default Vendor Code is used Value from parameter ‘ulVendorCode’ is used
			Bit 1: CONFIG MODEL TYPE DISABLED/ENABLED
		0 1	Default Model Type is used Value from parameter ‘ulModelType’ is used
			Bit 2: CONFIG SW VERSION DISABLED/ENABLED
		0 1	Default Software Version is used Value from parameter ‘ulSwVersion’ is used
ulWdgTime	UINT32		Watchdog supervision
		0 20... 65535	Watchdog supervision deactivated Watchdog time in milliseconds
ulSlaveStationAddr	UINT32		Slave Station Address

		1 .. 64 255	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 **Note:** 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
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 31: `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;      /* System flags   */
  TLR_UINT32    ulCcLinkFlags;      /* CC-Link Flags  */

  TLR_UINT32    ulWdgTime;          /* Watchdog time  */

  TLR_UINT32    ulSlaveStationAddr; /* Node ID        */
  TLR_UINT32    ulBaudRate;         /* BaudRate       */

  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: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error Codes CC-Link APS-Task on page 86.
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
structure CCLINK_APS_WARMSTART_CNF_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
ulCcLinkFlags	UINT32	 0 1 0 1 0 1	CC-Link Flags Bit 0: CONFIG VENDOR CODE DISABLED/ENABLED Default Vendor Code is used Value from parameter ‘ulVendorCode’ is used Bit 1: CONFIG MODEL TYPE DISABLED/ENABLED Default Model Type is used Value from parameter ‘ulModelType’ is used Bit 2: CONFIG SW VERSION DISABLED/ENABLED Default Software Version is used Value from parameter ‘ulSwVersion’ is used

ulWdgTime	UINT32		Watchdog supervision
		0 20 .. 65535	Watchdog supervision deactivated Watchdog time in milliseconds
ulSlaveStationAddr	UINT32		Slave Station Address
		1 .. 64 255	Slave Station Address from Packet Slave Station Address and Baudrate from rotary switches (Allowed for devices with rotary switches)
ulBaudRate	UINT32		Baudrate
		0 1 2 3 4	156 kBaud 625 kBaud 2500 kBaud 5 MBaud 10 MBaud
			Note: 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		Station Type
		0 1	Remote I/O Station Remote Device Station
ulNoStation	UINT32		Number of Occupied Stations
		1 1 ... 4	Range for Remote I/O Station Range for Remote Device Station
ulCcLinkVersion	UINT32		CC-Link Version
		0 1	Version 1 Version 2 (Remote Device Station only)
ulExtensionCycle	UINT32		Number of extension cycles for CC-Link Version 1
		0	Single setting / one cycle
			Number of extension cycles for CC-Link Version 2
		0 1 2 3	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		Behavior in case of bus error
		0 1	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 32: `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 33: CCLINK_APS_PCK_WARMSTART_CNF – Packet Status/Error

3.1.2 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 `RCX_REGISTER_APP_REQ`, code `0x2F10`, see reference [1].
- This request will be denied if the configuration lock flag is set

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	*	Mixed
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_8POINTS	0x00000002L	8	Mixed
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_16POINTS	0x00000003L	16	Mixed
CCLINK_SLAVE_IO_TYPES_POINTS_MIXED_32POINTS	0x00000004L	32	Mixed
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_DEP_ON_STATION	0x00000005L	*	Input
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_8POINTS	0x00000006L	8	Input
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_16POINTS	0x00000007L	16	Input
CCLINK_SLAVE_IO_TYPES_POINTS_INPUT_32POINTS	0x00000008L	32	Input
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_DEP_ON_STATION	0x00000009L	*	Output
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_8POINTS	0x0000000AL	8	Output
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_16POINTS	0x0000000BL	16	Output
CCLINK_SLAVE_IO_TYPES_POINTS_OUTPUT_32POINTS	0x0000000CL	32	Output
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_DEP_ON_STATION	0x0000000DL	*	Composite
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_8POINTS	0x0000000EL	8	Composite
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_16POINTS	0x0000000FL	16	Composite
CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_32POINTS	0x00000010L	32	Composite

Table 34: 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

/* Switch for address */
#define CCLINK_APS_SYS_FLAG_ADDRESS_SWITCH           0x00000010L
/* Switch for baud rate */
#define CCLINK_APS_SYS_FLAG_BAUD_SWITCH              0x00000020L

#define CCLINK_APS_WD_OFF                            0x00000000L
#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
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error Codes CC-Link APS-Task on page 86.
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
structure CCLINK_APS_SET_CONFIGURATION_REQ_DATA_T			
ulSystemFlags	UINT32		System Flags BIT 0: AUTOSTART / APPLICATION CONTROLLED
		0	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
		1	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: Reserved for further use, set to zero
			BIT 4: ADDRESS_SWITCH
			Should be set when hardware address switch is used and there is no TAG present.
			BIT 5: BAUD_SWITCH
			Should be set when hardware baudrate switch is used and there is no TAG present.
			BIT 6 - 31: Reserved for further use, set to zero
ulWdgTime	UINT32		Watchdog supervision
		0	Watchdog supervision deactivated
		20 .. 65535	Watchdog time in milliseconds

ulCcLinkFlags	UINT32		CC-Link Flags
			Bit 0: CONFIG VENDOR CODE DISABLED/ENABLED
		0 1	Default Vendor Code is used Value from parameter ‘ulVendorCode’ is used
			Bit 1: CONFIG MODEL TYPE DISABLED/ENABLED
		0 1	Default Model Type is used Value from parameter ‘ulModelType’ is used
			Bit 2: CONFIG SW VERSION DISABLED/ENABLED
		0 1	Default Software Version is used Value from parameter ‘ulSwVersion’ is used
ulSlaveStationAddr	UINT32		Slave Station Address
		1 .. 64 255	Slave Station Address from Packet Slave Station Address and Baudrate from rotary switches (On devices with rotary switches, only)
ulBaudRate	UINT32		Baudrate
		0 1 2 3 4	156 kBaud 625 kBaud 2500 kBaud 5 MBaud 10 MBaud
			Note: 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		Station Type
		0 1	Remote I/O Station Remote Device Station
ulNoStation	UINT32		Number of Occupied Stations
		1 1 .. 4	Range for Remote I/O Station Range for Remote Device Station
ulCcLinkVersion	UINT32		CC-Link Version
		0 1	Version 1 Version 2 (Remote Device Station only)
ulExtensionCycle	UINT32	0	Number of extension cycles for CC-Link Version 1 Single setting / one cycle
			Number of extension cycles for CC-Link Version 2
		0 1 2 3	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		Behavior in case of bus error
		0 1	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 35: `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: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error Codes CC-Link APS-Task on page 86.
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 36: CCLINK_APS_PCK_SET_CONFIGURATION_CNF_T – Set Warmstart Parameter Confirmation

3.1.3 CCLINK_APS_SET_DATA_LOOP_REQ/CNF – Set Data Loop

This packet enables or disables data loop operation. If the ulMode parameter is set to the value 1, data loop is enabled meaning the CC-Link Slave sends the unchanged data received from the master back to it.

This slave behavior continues until

- a channel init occurs
- a CCLINK_APS_SET_DATA_LOOP_REQ packet with ulMode parameter equal to 0 is sent to the CC-Link Slave.

This means, if data loop operation should still be continued after a channel init, a new CCLINK_APS_SET_DATA_LOOP_REQ packet with ulMode equal to 1 has to be sent

Packet Structure Reference

```
#define CCLINK_APS_SET_DATA_LOOP_REQ      0x000046E0L
#define CCLINK_APS_SET_DATA_LOOP_CNF      0x000046E1L

/** type of CCLINK_APS_SET_DATA_LOOP_REQ_DATA_Ttag */
typedef struct CCLINK_APS_SET_DATA_LOOP_REQ_DATA_Ttag
  CCLINK_APS_SET_DATA_LOOP_REQ_DATA_T;

#define CCLINK_APS_DATA_LOOP_DISABLE  0x00000000L
#define CCLINK_APS_DATA_LOOP_ENABLE   0x00000001L

 struct CCLINK_APS_SET_DATA_LOOP_REQ_DATA_Ttag
{
  TLR_UINT32 ulMode;
} ;

/** type of CCLINK_APS_SET_DATA_LOOP_REQ_Ttag */
typedef struct CCLINK_APS_SET_DATA_LOOP_REQ_Ttag
  CCLINK_APS_SET_DATA_LOOP_REQ_T;

 struct CCLINK_APS_SET_DATA_LOOP_REQ_Ttag
{
  TLR_PACKET_HEADER_T                tHead; /** packet header */
  CCLINK_APS_SET_DATA_LOOP_REQ_DATA_T tData; /** packet data   */
} ;
```

Packet Description

structure CCLINK_APS_SET_DATA_LOOP_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_CCLAPS	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 Status/Error Codes CC-Link APS-Task on page 86.
ulCmd	UINT32	0x46E0	CCLINK_APS_SET_DATA_LOOP_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
Structure CCLINK_APS_SET_DATA_LOOP_REQ_DATA_T			
ulMode	UINT32	0 1	Data loop operation is finished. Data loop operation is enabled.

Table 37: CCLINK_APS_SET_DATA_LOOP_REQ_T – Set Data Loop Request

Packet Structure Reference

```
#define CCLINK_APS_SET_DATA_LOOP_REQ      0x000046E0L
#define CCLINK_APS_SET_DATA_LOOP_CNF      0x000046E1L

/** type of CCLINK_APS_SET_DATA_LOOP_CNF_Ttag */
typedef struct CCLINK_APS_SET_DATA_LOOP_CNF_Ttag
  CCLINK_APS_SET_DATA_LOOP_CNF_T;

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

Packet Description

structure CCLINK_APS_SET_DATA_LOOP_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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		See section Status/Error Codes CC-Link APS-Task on page 86.
ulCmd	UINT32	0x46E1	CCLINK_APS_SET_DATA_LOOP_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change

Table 38: CCLINK_APS_SET_DATA_LOOP_CNF_T– Confirmation to Set Data Loop Request

3.2 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 39: 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_SENDPACKET_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:

Packet	Command code (REQ/CNF or IND/RES)	Page
CCLINK_SLAVE_INITIALIZE_REQ/CNF – Initialization of CC-Link Slave	0x4500/0x4501	40
CCLINK_SLAVE_REGISTER_REQ/CNF – Register Application	0x4502/0x4503	42
CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle	0x4504/0x4505	45
CCLINK_SLAVE_SET_BUSPARAM_REQ/CNF – Set Bus Parameters	0x4506/0x4507	48
CCLINK_SLAVE_STARTSTOP_REQ/CNF – Start/Stop Communication with Network	0x4508/0x4509	53
CCLINK_SLAVE_GET_CCL_STATUS_REQ/CNF – Get CC-Link Status	0x450A/0x450B	56
CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ/CNF – Change CC-Link Slave Status	0x450C/0x450D	59
CCLINK_SLAVE_GET_BUS_PARAM_REQ/CNF – Get Bus Parameters	0x450E/0x450F	63
CCLINK_SLAVE_STATE_CHANGE_IND/RES – Change of State Indication	0x451E/0x451F	65
CCLINK_SLAVE_SET_WATCHDOG_FAIL_REQ/CNF – Set Watchdog Fail	0x45AA/0x45AB	70
CCLINK_SLAVE_STATE_CHANGE_IND/RES – Change of State Indication	0x4510/0x4511	72

Table 40: Overview over the Packets of the CC-Link Slave-Task of the CC-Link Slave Protocol Stack

3.2.1 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.

> **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 CCLINK_SLAVE_PACKET_INITIALIZE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4500	CCLINK_SLAVE_INITIALIZE_REQ – Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
Structure CCLINK_SLAVE_INITIALIZE_REQ_DATA_T			
ulReserved	UINT32		Reserved for further use, set to zero

Table 41: 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 CCLINK_SLAVE_PACKET_INITIALIZE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4501	CCLINK_SLAVE_INITIALIZE_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
Structure CCLINK_SLAVE_INITIALIZE_CNF_DATA_T			
ulReserved	UINT32		Reserved for further use, set to zero

Table 42: CCLINK_SLAVE_PACKET_INITIALIZE_CNF_T – Initialization of CC-Link Slave Confirmation

3.2.2 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 RCX_REGISTER_APP_REQ, code 0x2F10, see reference [1].

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 CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
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 44: CCLINK_SLAVE_REGISTER_REQ – Packet Status/Error.*
ulCmd	UINT32	0x4502	CCLINK_SLAVE_REGISTER_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
structure CCLINK_SLAVE_APP_REGISTER_REQ_DATA_T			
ulReserved	UINT32		Reserved for further use, set to zero

Table 43: CCLINK_SLAVE_PACKET_APP_REGISTER_REQ_T – Register Application Request

Packet Status/Error

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

Table 44: 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 CCLINK_SLAVE_PACKET_APP_REGISTER_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See *Table 46: CCLINK_SLAVE_REGISTER_CNF – Packet Status/Error.*
ulCmd	UINT32	0x4503	CCLINK_SLAVE_REGISTER_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
Structure CCLINK_SLAVE_APP_REGISTER_CNF_DATA_T			
ulReserved	UINT32		Reserved for further use, set to zero

Table 45: 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 46: CCLINK_SLAVE_REGISTER_CNF – Packet Status/Error

3.2.3 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

```
/** type of <code>CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_Ttag</code> */
typedef struct CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_Ttag
  CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_T;

#define CCLINK_SLAVE_MAX_BIT_DATA         112
#define CCLINK_SLAVE_MAX_REGISTER_DATA    256

#define CCLINK_SLAVE_SEND_RECV_BUFFER_SIZE \
  (CCLINK_SLAVE_MAX_BIT_DATA + CCLINK_SLAVE_MAX_REGISTER_DATA)

struct CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_Ttag
{
  TLR_UINT32 ulReserved;
};
```

Packet Description

structure CCLINK_SLAVE_PACKET_GET_BUFFER_HANDLE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4504	CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
Structure CCLINK_SLAVE_GET_BUFFER_HANDLE_REQ_DATA_T			
ulReserved	UINT32	0	Reserved, set to zero

Table 47: 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 CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4505	CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
Structure CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_DATA_T			
ulReserved	UINT32		Reserved for further use
ulRecvBuffer	UINT32		Receive Buffer
ulSendBuffer	UINT32		Send Buffer

Table 48: CCLINK_SLAVE_GET_BUFFER_HANDLE_CNF_T – Get Buffer Handle Confirmation

3.2.4 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.

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  0x00000001L
#define CCLINK_SLAVE_MAX_SLAVE_STATION_ADDR  0x00000040L

#define CCLINK_SLAVE_BAUD_156K          0x00000000L   /* 156kBaud  */
#define CCLINK_SLAVE_BAUD_625K          0x00000001L   /* 625kBaud  */
#define CCLINK_SLAVE_BAUD_2500K         0x00000002L   /* 2500kBaud */
#define CCLINK_SLAVE_BAUD_5M            0x00000003L   /* 5MBaud    */
#define CCLINK_SLAVE_BAUD_10M           0x00000004L   /* 10MBaud   */
#define CCLINK_SLAVE_MAX_BAUD           CCLINK_SLAVE_BAUD_10M

#define CCLINK_SLAVE_STATION_TYPE_IO_DEVICE     0x00000000L
#define CCLINK_SLAVE_STATION_TYPE_REMOTE_DEVICE 0x00000001L

#define CCLINK_SLAVE_NO_STATION_IO_DEVICE       0x00000001L

#define CCLINK_SLAVE_LIMITED_SLAVE_STATION_ADDR   0x00000020L

#define CCLINK_SLAVE_MIN_NO_STATION_REMOTE_DEVICE 0x00000001L
#define CCLINK_SLAVE_MAX_NO_STATION_REMOTE_DEVICE 0x00000004L

#define CCLINK_SLAVE_CCLINK_VERSION_1   0x00000000L
#define CCLINK_SLAVE_CCLINK_VERSION_2   0x00000001L

#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_SINGLE    0x00000000L
#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_DOUBLE    0x00000001L
#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_QUADRUPLE 0x00000002L
#define CCLINK_SLAVE_CCLINK_EXTENSION_CYCLE_OCTUPLE   0x00000003L

#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

#define CCLINK_SLAVE_MAX_IO_TYPES_POINTS  CCLINK_SLAVE_IO_TYPES_POINTS_COMPOSITE_32POINTS

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    ulIoTypesPoints;
  TLR_BOOLEAN32 fHoldClrMode;
};

typedef struct CCLINK_SLAVE_ADD_PARAM_Ttag
  CCLINK_SLAVE_ADD_PARAM_T;

#define CCLINK_SLAVE_MAX_VENDOR_CODE    0x0000FFFFL
#define CCLINK_SLAVE_MAX_MODEL_TYPE     0x000000FFL
#define CCLINK_SLAVE_MAX_SW_VERSION     0x0000003FL

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		Baudrate
		0	156 kBaud
		1	625 kBaud
		2	2500 kBaud
		3	5 MBaud
		4	10 MBaud
ulStationType	UINT32		Station Type
		0	Remote I/O Station
		1	Remote Device Station
ulNoStation	UINT32		Number of occupied stations
		1	Range for Remote I/O Station
		1 .. 4	Range for Remote Device Station
ulCcLinkVersion	UINT32		Version of CC-Link protocol
		0	Version 1
		1	Version 2 (Remote Device Station only)
ulExtensionCycle	UINT32		Number of extension cycles for CC-Link Version 1
		0	Single setting / one cycle
			Number of extension cycles for CC-Link Version 2
		0	Single setting / one cycle
		1	Double setting / two cycles
		2	Quadruple setting / four cycles
		3	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		Behavior in case of bus error
		0	Clear output data
		1	Hold last received output data

Table 49: 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 50: 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 CCLINK_SLAVE_SET_BUSPARAM_REQ_DATA_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
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	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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4506	*CCLINK_SLAVE_SET_BUS_PARAM_REQ* - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
Structure CCLINK_SLAVE_SET_BUSPARAM_REQ_DATA_T			
CCLINK_ SLAVE_CFG_ BUS_PARAM_T	*See Table 49: CCLINK_SLAVE_CFG_BUS_PARAM_T - Bus Parameter*		
CCLINK_ SLAVE_CFG_ ADD_PARAM_T	*See Table 50: CCLINK_SLAVE_CFG_ADD_PARAM_T - Additional Configuration*		

Table 51: 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 CCLINK_SLAVE_PACKET_SET_BUSPARAM_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4507	CCLINK_SLAVE_SET_BUSPARAM_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
Structure CCLINK_SLAVE_SET_BUSPARAM_CNF_DATA_T			
CCLINK_ SLAVE_CFG_ BUS_PARAM_T	*See Table 49: CCLINK_SLAVE_CFG_BUS_PARAM_T - Bus Parameter*		
CCLINK_ SLAVE_CFG_ ADD_PARAM_T	*See Table 50: CCLINK_SLAVE_CFG_ADD_PARAM_T - Additional Configuration*		

Table 52: CCLINK_SLAVE_PACKET_SET_BUSPARAM_CNF_T –Set Bus Parameter Confirmation

3.2.5 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 has been replaced by packet RCX_START_STOP_COMM_REQ/CNF which contains identical functionality.

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 CCLINK_SLAVE_PACKET_STARTSTOP_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4508	CCLINK_SLAVE_STARTSTOP_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
Structure CCLINK_SLAVE_STARTSTOP_REQ_DATA_T			
ulMode	UINT32	 0 1	Depending on this assignment, communication is either started or stopped: Stop communication with Network Start communication with Network

Table 53: 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 CCLINK_SLAVE_PACKET_STARTSTOP_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4509	CCLINK_SLAVE_STARTSTOP_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
structure CCLINK_SLAVE_STARTSTOP_CNF_DATA_T			
ulMode	UINT32	 0 1	Depending on this assignment, communication is either started or stopped: Stop communication with network Start communication with network

Table 54: CCLINK_SLAVE_PACKET_STARTSTOP_CNF_T – Start/Stop Communication Confirmation

3.2.6 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 CCLINK_SLAVE_PACKET_GET_CCL_STATUS_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x450A	CCLINK_SLAVE_GET_CCL_STATUS_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
structure CCLINK_SLAVE_GET_CCL_STATUS_REQ_DATA_T			
ulReserved	UINT32		Reserved for further use, set to zero

Table 55: 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         0x00000001L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK 0x00000002L
#define CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK               0x00000004L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK         0x00000008L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK   0x00000010L
#define CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK              0x00000060L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK           0x00000080L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK   0x00000F00L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK  0x0000F000L

#define CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK                    0x00000001L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVAL_NUM_OF_POINTS_MSK 0x00000002L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK             0x00000004L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK           0x00000008L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK        0x00000010L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK           0x00000020L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK                           0x00000040L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK                      0x00000080L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK          0x00000100L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK    0x00000200L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK                 0x00000400L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK                           0x00000800L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK      0x00001000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK               0x00002000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK         0x0000C000L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SINGLE_MSK          0x00000000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_DOUBLE_MSK          0x00004000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_QUADRUPLE_MSK       0x00008000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_OCTUPLE_MSK         0x0000C000L

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 CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x450B	CCLINK_SLAVE_GET_CCL_STATUS_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
structure CCLINK_SLAVE_GET_CCL_STATUS_CNF_DATA_T			
ulReserved	UINT32		Reserved for further use
ulMasterState	UINT32		See page above.
ulSlaveState	UINT32		See page above.

Table 56: CCLINK_SLAVE_PACKET_GET_CCL_STATUS_CNF_T – Get CC-Link Status Confirmation

3.2.7 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      0x00000001L
#define CCLINK_SLAVE_EVAL_SWITCH_CHANGE_PARAM_MSK   0x00000002L
#define CCLINK_SLAVE_EVAL_WATCHDOG_ERROR_PARAM_MSK  0x00000004L

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 CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
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		See section Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x450C	CCLINK_SLAVE_CHANGE_SLAVE_STATUS_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
structure CCLINK_SLAVE_CHANEGE_SLAVE_STATUS_REQ_DATA_T			
ulFlags	UINT32		Flags Bit 0:Parameter fFuseStatus will be evaluated if this bit is set Bit 1:Parameter fSwitchChange will be evaluated if this bit is set Bit 2:Parameter fWatchdogError 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 ulFlags 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 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 corresponding bit in parameter ulFlags is set Clear watchdog error Set watchdog error

Table 57: 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 CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x450D	CCLINK_SLAVE_CHANGE_SLAVE_STATUS_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
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 58: CCLINK_SLAVE_PACKET_CHANGE_SLAVE_STATUS_CNF_T – Change CC-Link Slave Status Confirmation

3.2.8 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 CCLINK_SLAVE_PACKET_GET_BUS_PARAM_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
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	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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x450E	CCLINK_SLAVE_GET_BUS_PARAM_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information

Table 59: 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 CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x450F	CCLINK_SLAVE_GET_BUS_PARAM_CNF - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change
structure CCLINK_SLAVE_GET_BUS_PARAM_CNF_DATA_T			
CCLINK_ SLAVE_CFG_ BUS_PARAM_T	*See Table 49: CCLINK_SLAVE_CFG_BUS_PARAM_T - Bus Parameter*		
CCLINK_ SLAVE_CFG_ ADD_PARAM_T	*See Table 50: CCLINK_SLAVE_CFG_ADD_PARAM_T - Additional Configuration*		

Table 60: CCLINK_SLAVE_PACKET_GET_BUS_PARAM_CNF_T – Get Bus Parameter Confirmation

3.2.9 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 3.2.2 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;

struct CCLINK_SLAVE_STATE_CHANGE_IND_DATA_Ttag
{
  CCLINK_SLAVE_SLAVE_STATE_T     tSlaveState;
  CCLINK_SLAVE_EXTENDED_STATE_T  tExtendedState;

};

typedef struct CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_Ttag
  CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_T;

struct CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_Ttag
{
  TLR_PACKET_HEADER_T                  tHead;
  CCLINK_SLAVE_STATE_CHANGE_IND_DATA_T tData;

};
```

Packet Description

structure CCLINK_SLAVE_PACKET_STATE_CHANGE_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x451E	CCLINK_SLAVE_STATE_CHANGE_IND - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
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 61: 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        0x00000001L
#define CCLINK_SLAVE_STATE_FLAG_RUN        0x00000002L
#define CCLINK_SLAVE_STATE_FLAG_COM        0x00000004L
#define CCLINK_SLAVE_STATE_FLAG_BUS_ON     0x00000008L
#define CCLINK_SLAVE_STATE_FLAG_COMM_ERROR 0x00000010L

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 0x00000001L
#define CCLINK_SLAVE_EXT_STATE_CTRL     0x00000002L
#define CCLINK_SLAVE_EXT_STATE_NRDY     0x00000004L

#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_MSK         0x00000001L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK 0x00000002L
#define CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK               0x00000004L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK         0x00000008L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK   0x00000010L
#define CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK              0x00000060L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK           0x00000080L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK   0x00000F00L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK  0x0000F000L

#define CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK                    0x00000001L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVAL_NUM_OF_POINTS_MSK 0x00000002L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK             0x00000004L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK           0x00000008L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK        0x00000010L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK           0x00000020L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK                           0x00000040L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK                      0x00000080L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK          0x00000100L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK    0x00000200L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK                 0x00000400L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK                           0x00000800L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK      0x00001000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK               0x00002000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK         0x0000C000L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SINGLE_MSK          0x00000000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_DOUBLE_MSK          0x00004000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_QUADRUPLE_MSK       0x00008000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_OCTUPLE_MSK         0x0000C000L

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 extended slave state is described in detail in section *Extended Status* on page 76.

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 CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x451F	CCLINK_SLAVE_STATE_CHANGE_RES - Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change

Table 62: CCLINK_SLAVE_PACKET_STATE_CHANGE_RES_T – Change of State Response

3.2.10 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 CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
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	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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x45AA	CCLINK_SLAVE_SET_WATCHDOG_FAIL_REQ - Command
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information

Table 63: 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 CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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		See section Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x45AB	CCLINK_SLAVE_SET_WATCHDOG_FAIL_CNF– Command
ulExt	UINT32		Extension, reserved
ulRout	UINT32		Routing Information, do not change

Table 64: CCLINK_SLAVE_PACKET_SET_WATCHDOG_FAIL_CNF_T – Set Watchdog Fail Confirmation

3.2.11 CCLINK_SLAVE_STATE_CHANGE_IND/RES – Change of State Indication

This packet can be used to retrieve the currently configured values of the counts of receive data (Variable `ulRecvDataCnt` of confirmation packet) and send data (Variable `ulSendDataCnt` of confirmation packet) in CC-Link data communication.

Packet Structure Reference

```
/*****************************************************************************/
/** type of <code>CCLINK_SLAVE_PACKET_GET_IO_INFO_REQ_Ttag</code> */
typedef struct CCLINK_SLAVE_PACKET_GET_IO_INFO_REQ_Ttag
  CCLINK_SLAVE_PACKET_GET_IO_INFO_REQ_T;

struct CCLINK_SLAVE_PACKET_GET_IO_INFO_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;                  /** packet header.            */
};

/*****************************************************************************/
```

Packet Description

structure CCLINK_SLAVE_PACKET_GET_IO_INFO_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_CCLSLAVE	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	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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4510	CCLINK_SLAVE_GET_IO_INFO_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 65: CCLINK_SLAVE_GET_IO_INFO_REQ – Get I/O Info Request

Packet Structure Reference

```
/*****************************************************************************/
/** type of <code>CCLINK_SLAVE_GET_IO_INFO_CNF_DATA_Ttag</code> */
typedef struct CCLINK_SLAVE_GET_IO_INFO_CNF_DATA_Ttag
  CCLINK_SLAVE_GET_IO_INFO_CNF_DATA_T;

struct CCLINK_SLAVE_GET_IO_INFO_CNF_DATA_Ttag
{
  TLR_UINT32 ulSendDataCnt;     /* Send data count    */
  TLR_UINT32 ulRecvDataCnt;     /* Receive data count */
};

/*****************************************************************************/
/** type of <code>CCLINK_SLAVE_PACKET_GET_IO_INFO_CNF_Ttag</code> */
typedef struct CCLINK_SLAVE_PACKET_GET_IO_INFO_CNF_Ttag
  CCLINK_SLAVE_PACKET_GET_IO_INFO_CNF_T;

struct CCLINK_SLAVE_PACKET_GET_IO_INFO_CNF_Ttag
{
  TLR_PACKET_HEADER_T                 tHead; /** packet header.            */
  CCLINK_SLAVE_GET_IO_INFO_CNF_DATA_T tData; /** packet confirmation data. */
};

/*****************************************************************************/
```

Packet Description

structure CCLINK_SLAVE_PACKET_GET_IO_INFO_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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	8	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 Status/Error codes CC-Link Slave-Task on page 88.
ulCmd	UINT32	0x4511	CCLINK_SLAVE_GET_IO_INFO_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure CCLINK_SLAVE_GET_IO_INFO_CNF_DATA_T			
ulSendDataCnt	UINT32	0 ... 2^{32}-1	Send Data Count
ulRecvDataCnt	UINT32	0 ... 2^{32}-1	Receive Data Count

Table 66: CCLINK_SLAVE_GET_IO_INFO_CNF – Confirmation to Get I/O Info Request

3.3 Hardware Switches for the Adjustment of Slave Address and Baudrate

For handling address and baud rate switches on a netX device, the firmware must be enabled to evaluate the switch values from the hardware. This can be done by setting a special TAG via the “Tag List Editor” tool. In this case, the values which are configured via database or packet interface will be ignored.

If the hardware switch function is not enabled via TAG, then the firmware uses the values set either via NXD or IniBatch database or via packet interface (SET_CONFIGURATION_REQ or RCX_SET_HW_SWITCH_VALUE_REQ)

Enabling and disabling Address and Baudrate Switching

On database files and SET_CONFIGURATION_REQ evaluating the switches can be activated or deactivated. This information is located at the System Flags

CCLINK_APS_SYS_FLAG_ADDRESS_SWITCH = 0x10

CCLINK_APS_SYS_FLAG_BAUD_SWITCH = 0x20.

Also see section CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration on page 30.

Behavior at Start-up

In general, the firmware stack can be configured in different ways. Only one type of configuration can be active at a certain time. These are evaluated at start-up in the following order:

- SYCON.net database
- iniBatch database (via netX Configuration Tool)
- Warmstart Request packet (compatibility)
- Set Configuration Request packet

After a Restart the stack will first search for the SYCON.net database files (config.nxd). If these are found all other configuration methods will not be accepted. If no SYCON.net database exists, but an iniBatch database exists, its configuration will be used and configuration packets will be not accepted.

If no database is found the stack is unconfigured until the receipt of the first configuration packet. In this case the firmware waits for the SET_CONFIGURATION_REQ or WARMSTART_REQ packet. Once one of these packets (i.e. SET_CONFIGURATION_REQ) was received, the other one (i.e. WARMSTART_REQ) will be rejected.

The host has the possibility to modify the configuration with the packet RCX_SET_FW_PARAMETER_REQ/CNF (Set the Value of the Firmware Parameter).

Using the hardware switches for adjusting of slave address and baudrate requires the option *Application_controlled* being active (either when configuring using SET_CONFIGURATION_REQ packet (see CCLINK_APS_SET_CONFIGURATION_REQ/CNF – Set Configuration on page 30) or in the SYCON.net database file config.nxd).

The stack will start the device with the received configuration as soon as the application ready flag is set by the host application.

On starting the stack the hardware switches are evaluated if hardware switches are enabled via the TAG. The values from the hardware switches will overwrite the values, which was set via database or packet previously. This can be avoided if the hardware switches are disabled via the “Tag List Editor” tool. A description of the “Tag List Editor” tool is given in reference #3.).

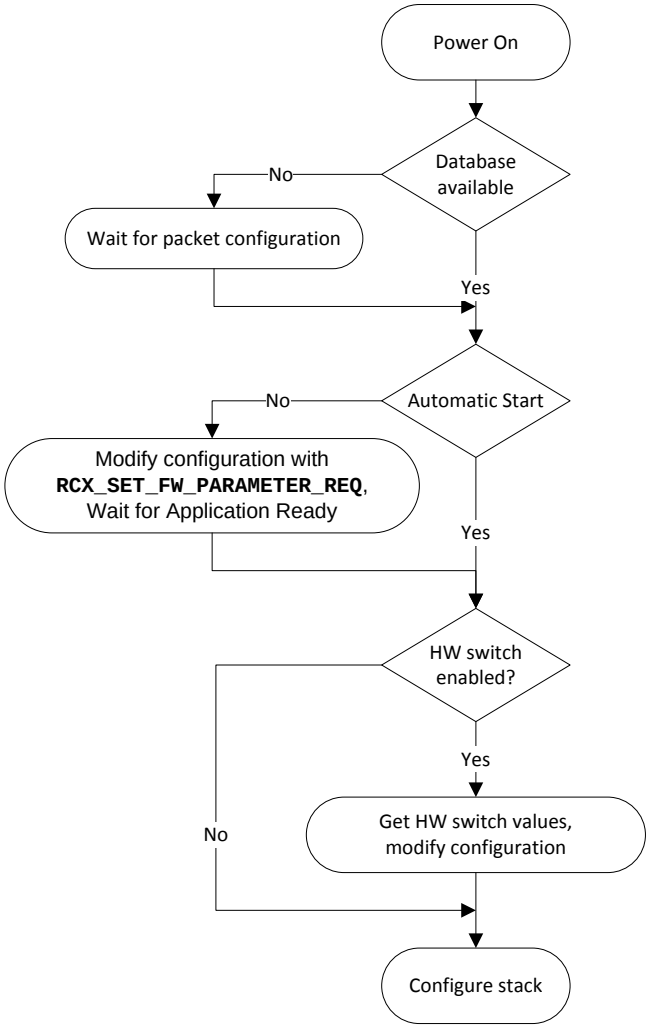

Figure 2: Start-up Process

4 CC-Link Status Information

4.1 Common 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 reference [1]).

4.2 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 reference [1]).

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 0x00000001L
#define CCLINK_SLAVE_EXT_STATE_CTRL     0x00000002L
#define CCLINK_SLAVE_EXT_STATE_NRDY     0x00000004L

#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_MSK         0x00000001L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MAS_STAT_USER_APP_PRG_ERR_CHK_MSK 0x00000002L
#define CCLINK_SLAVE_CCL_MASTER_ST1_REFRESH_STARTUP_MSK               0x00000004L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_STATUS_MSK         0x00000008L
#define CCLINK_SLAVE_CCL_MASTER_ST1_TRANSIENT_DATA_RECEPTION_EN_MSK   0x00000010L
#define CCLINK_SLAVE_CCL_MASTER_ST1_PROTOCOL_VERSION_MSK              0x00000060L
#define CCLINK_SLAVE_CCL_MASTER_ST1_MASTER_STATION_TYPE_MSK           0x00000080L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RY_INFO_TRANSMISSION_POINTS_MSK   0x00000F00L
#define CCLINK_SLAVE_CCL_MASTER_ST2_RWW_INFO_TRANSMISSION_POINTS_MSK  0x0000F000L

#define CCLINK_SLAVE_CCL_SLAVE_ST1_FUSE_STATUS_MSK                    0x00000001L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_UNIT_ERROR_INVAL_NUM_OF_POINTS_MSK 0x00000002L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_REFRESH_RECEIVE_MSK             0x00000004L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_NO_PARAMETER_RECEIVE_MSK           0x00000008L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_SWITCH_CHANGE_DETECTION_MSK        0x00000010L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_CYCLIC_COMMUNICATION_MSK           0x00000020L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_RES1_MSK                           0x00000040L
#define CCLINK_SLAVE_CCL_SLAVE_ST1_WDT_ERROR_MSK                      0x00000080L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_STATUS_MSK          0x00000100L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_DATA_RECEPTION_EN_MSK    0x00000200L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSIENT_TYPE_MSK                 0x00000400L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES2_MSK                           0x00000800L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_TRANSMISSION_ROUTE_STATUS_MSK      0x00001000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_RES_FIXED_TO_ONE_MSK               0x00002000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SETTING_MSK         0x0000C000L

#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_SINGLE_MSK          0x00000000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_DOUBLE_MSK          0x00004000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_QUADRUPLE_MSK       0x00008000L
#define CCLINK_SLAVE_CCL_SLAVE_ST2_EXTENDED_CYCLE_OCTUPLE_MSK         0x0000C000L

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;

  TLR_UINT32 ulSqErrorCount;

  TLR_UINT32 ulCommErrorCnt;
  TLR_UINT32 ulLastCommError;
  TLR_UINT32 ulAddCommErrorInfo;
};
```

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 67: 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 68: 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 69: 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 70: 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 71: 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 72: 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 73: 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 74: 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_INVAL_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 75: 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 76: 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 Data link is aborted.

Table 77: 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 Data link is aborted.

Table 78: 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 79: 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 Data link is continued.

Table 80: 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 81: 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 Data link is continued.

Table 82: 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	RWw information transmission points, see below

Table 83: 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 84: 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 85: 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 86: 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 87: 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 88: 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 89: 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 90: 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 91: 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.

ulSqErrorCount

This entry contains the count of sequence errors.

ulCommErrorCnt

This entry contains the count of communication errors.

ulLastCommError

This entry contains the error code of the last communication error that occurred.

ulAddCommErrorInfo

This entry contains the additional info concerning the last communication error that occurred.

5 Status/Error Codes Overview

5.1 Status/Error Codes CC-Link APS-Task

Hexadecimal Value	Definition Description
0x0	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_INVALD_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.

Hexadecimal Value	Definition Description
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 92: Status/Error Codes CC-Link APS-Task

5.2 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.

Hexadecimal Value	Definition Description
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 93: Status/Error Codes CC-Link Slave-Task

6 Appendix

6.1 List of Figures

6.2 List of Tables

6.3 Contacts

Headquarters

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

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

Subsidiaries

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

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

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

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

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

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

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

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

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

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

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

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

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

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