Protocol API

Sercos Slave

V3.5.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC100205API17EN | Revision 17 | English | 2017-08 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This manual describes the packet interface of the Sercos slave device implementation version 3 on the netX chip. The aim of this manual is to support the integration of devices based on the netX chip into own applications based on driver functions or direct access to the dual-port memory.

The general mechanism of data transfer, for example how to send and receive a packet or how to perform a warmstart is independent from the protocol. These procedures are common to all devices and are described in the '*netX DPM Interface manual*'.

The term "Sercos" in the context of this manual always means "Sercos in the third generation" if not otherwise indicated.

1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
15	2016-09-26	RG/DJ	-	Firmware/Stack version 3.4.0
			1.5	Changed maximum input and output lengths for different types of netX
			0	Added missing task queue names
			2.1	*Figure 1: Task Structure of the Sercos Slave Stack* has been updated
			5	Revised structure of chapter 5"*The Application Interface*"
			5.3.1	Description about parameter usSCP_FixCfg_OutputDataSize and usSCP_FixCfg_InputDataSize updated.
			5.3.1	Figure 29 updated.
			7.3	Error codes `0xC04E0055L` and `0xC04E0056L` added.
16	2017-07-17	DJ	-	Firmware/Stack version 3.5.0
			1.5	Supported Sercos Version: Changed to V1.1.2 and V1.3.1.
			5.3.1	DHCP can be used now and activated by ulTcpFlag.
			5.3.1	New value "NRTPC v2" added to bSCP_NRT_Version.
			5.3.1	New parameter: bSercosVersion to configure the stack working according V1.1.2 or V1.3.1. (Parameter bSercosVersion was a reserved parameter before.)
			5.6.3	Section *Remove Diagnostic Request*: Table 57 updated.
			5.9.2	Section *Check IP Parameters*: Figure 64 updated.
			-	General sections *Fundamentals* and *Dual-Port Memory* removed.
			4, 6	Section *Status Information* and *Topics for programmers using Linkable Object Modules* added.
			8.2	Section *IDNs created by the Stack*:
			8.5	Section *Migration from Stack Version V3.1 to V3.2* extended.
			8.8	Section *Migration from Stack Version V3.4 to V3.5* added.
17	2017-08-11	DJ, HH	-	Firmware/Stack version 3.5.0
			5.4.2	Section *Register for IDN read or write access*: Value for ulCmd corrected.
			5.5.10	Section *Unregister IDN Notify Request* added.
			5.5.11	Section *Unregister IDN Notify Confirmation* added.
			5.9.1, 5.9.3	Sections *Register check IP parameter* and *Activate Network Settings*: ulLen and figures corrected.
			8.2	IDN S-0-1042 removed.

Table 1: List of Revisions

1.3 Terms, Abbreviations and Definitions

Term	Description
xMAC	x Medium-Access Controller
xPEC	x Protocol-Execution-Controller
ARM	32 Bit CPU, part of the netX
Sercos	Serial Realtime Communication System
CP	SERCOS Communication Phase
AP (-task)	Application (-task) on top of the stack
HP	Hot-plug
DPM	Dual Port Memory
SVC	Service Channel
OD	Object Dictionary
IDN	Identification Number
Con_Clk	Controller Clock
Div_Clk	Divided Control Clock
NRT	Non real-time (state or mode)
UCC	Unified Communication Channel (UC Channel)
MDT	Master Data Telegram
AT	Acknowledge Telegram
S/IP	Sercos/IP Service
t_1	AT0 transmission starting time
t_3	Command value valid time
t_4	Synchronization time (T_{Sync})
t_6	UC transmission time - Begin of UC channel
t_7	UC transmission time - End of UC channel

Table 2: Terms, Abbreviations and Definitions

All variables, parameters, and data used in this manual have the LSB/MSB (“Intel”) data format. This corresponds to the convention of the Microsoft C Compiler. Only the Little-Endian mode is used for Sercos.

1.4 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] Hilscher Gesellschaft für Systemautomation mbH: Driver Manual, cifX Device Driver, Windows 2000/XP/Vista/7/CE V1.0.x.x. Revision 15, English, 2010

[3] IEC 61158-1 .. 6 Type 16 & Type 19

[4] IEEE 802.3, 2000

[5] Hilscher Gesellschaft für Systemautomation mbH: Ethernet Protocol API,. Revision 5, English, 2009

[6] Hilscher Gesellschaft für Systemautomation mbH: Application Note: Functions of the Integrated Web Server, Revision 4, English, 2012

[7] Sercos International: Communication Specification, Version 1.3.1-1.12: https://wiki.Sercos-service.org/latest-sys/wiki/images/latest-images/7/72/Communication_Protocol_V1.3.1-1.12.pdf

[8] Sercos International: Generic Device Profile, Version 1.3.1-1.3: https://wiki.Sercos-service.org/latest-sys/wiki/images/latest-images/3/38/Generic_Device_Profile_V1_3.1-1.3.pdf

[9] Sercos International: Internet protocol services, Version 1.3.1-1.2: https://wiki.Sercos-service.org/latest-sys/wiki/images/latest-images/4/4f/Internet_Protocol_Services_V1.3.1-1.2.pdf

[10] Sercos International: Function Specific Profile IO, Version 1.3.1-1.6: https://wiki.Sercos-service.org/latest-sys/wiki/images/latest-images/f/f7/Function_specific_Profile_IO_V1.3.1-1.6.pdf

[11] Sercos International: Function Specific Profile Drive, Version 1.3.1-2.16: https://wiki.Sercos-service.org/latest-sys/wiki/images/latest-images/5/56/Function_specific_Profile_Drives_V1.3.1-2.16.pdf

[12] Sercos International: Drive Classes, Version 1.3.1-11: https://wiki.Sercos-service.org/latest-sys/wiki/images/latest-images/a/a7/Drive-Classes_V1.3.1-1.1.pdf

Table 3: References to Documents

Note: To view the Sercos specifications you need to request an account at: https://accredit.Sercos-service.org/accredit/en/register

An overview of all Sercos Specifications can be found here: https://wiki.Sercos-service.org/latest/Documents

1.5 Technical Data

The data below applies to Sercos slave firmware and stack version V3.5.0. The firmware and stack has been written in order to meet the Sercos V1.3 specification.

Features	Value
Firmware/stack available for netX	netX 50, netX 51, netX 52, netX 100, netX 500
netX 50, netX 51, netX 52: Max. number of cyclic produced data (Tx) of all slaves	284 bytes (including Connection Control and IO Status)
netX 50, netX 51, netX 52: Max. number of cyclic consumed data (Rx) of all slaves	276 bytes (including Connection Control and IO Control)
netX 100, netX 500: Maximum number of cyclic produced (Tx) of all slaves	132 bytes (including Connection Control and IO Status)
netX 100, netX 500: Maximum number of cyclic consumed data (Rx) of all slaves	124 bytes (including Connection Control and IO Control)
Maximum number of slave devices	netX 52: 1 All others: 8
Maximum number of applicable Sercos addresses	1 … 511
Minimum cycle time	250 µs
Topology	Line and ring
Communication phases	NRT, CP0, CP1, CP2, CP3, CP4, HP0, HP1, HP2
Descriptors for connection (including Connection Control and IO Status/Control)	max. 64
Cross communication (CC)	supported
Acyclic Communication (Service Channel)	Read/Write/Standard Commands
Baud rate	100 MBit/s
Data transport layer	Ethernet II, IEEE 802.3
Supported Sercos version	Communication Specification Version 1.1.2 and V1.3.1
Supported Sercos Communication Profiles	SCP_FixCFG Version 1.1.1 SCP_VarCFG Version 1.1.1 SCP_VarCFG Version 1.1.3 SCP_HP Version 1.1.1 SCP_SysTime Version1.3
Supported User SCP Profiles	SCP_WD Version 1.1.1 SCP_Diag Version 1.1.1 SCP_RTB Version 1.1.1 SCP_Mux Version 1.1.1 SCP_Sig 1.1.1 SCP_ExtMuX V1.1.2 SCP_RTBListProd V1.3 SCP_RTBListCons V1.3 SCP_RTBWordProd V1.3 SCP_RTBWordCons V1.3 SCP_OvSBasic V1.3 SCP_WDCon V1.3
Supported FSP profiles	FSP_IO FSP_Drive FSP_Encoder
SCP Sync support	yes
SCP_NRTPC support	yes
S/IP support	yes

Features	Value
Identification LED feature supported	yes
Storage location of object dictionary	mixed mode
Configuration by	Packet to transfer configuration parameters SYCON.net with `config.nxd` or netX Configuration Tool with `inibatch.nxd`.

Table 4: Technical Data of the Protocol Stack

Limitations

- Modifications of the Service-Channel Object Dictionary are volatile after reset (if resides on device)
- UCC Communication: Second DPM Channel (“Ethernet Interface Task”) is not available on all hardware
- Clock parameters `ulDTDivClk` (Contents of DTDivClk register) and `ulDivClkLength` (DivClk Length) are no longer adjustable (since V3.1). `ulDivClkLength` is now always set to the fixed value 1 µs. In Div_Clk mode 0 it is not possible any more to configure aperiodic signals. ulConClkLength has now a maximum of 655350 ns.

Firmware Features

Feature	Integrated HTTP Server	Integrated TCP/IP Stack	Integrated Diagnostic and Remote Access via		
Target			Serial	USB	TCP
cifX	No	No	No	No	No
comX	Yes	Yes, but no packet interface via Dual-port memory	No	Yes	No
netJACK					
netX 50	Yes	Yes	No	No	No
netX 100	Yes	Yes	No	No	No
netX 500	Yes	Yes	No	No	No
NXIO 50	No	No	No	No	No
NXIO 100	No	No	No	No	No

Table 5: Technical Data of Firmware Features

Features of the integrated HTTP Server

- Firmware Identification
- Firmware Update
- Device Reset

See reference [6] for more details.

1.6 Legal Notes

Copyright

Important notes

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

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

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

Liability disclaimer

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

damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

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

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

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

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

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

Warranty

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

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

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

Costs of support, maintenance, customization and product care

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

Additional guarantees

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

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

Confidentiality

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

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

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

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

Export provisions

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

2 Getting started

This chapter gives an overview where to find some important information for starters. It also explains the parameters of the Sercos Slave firmware and the different ways how you can set them.

2.1 Task Structure of the Sercos Slave Stack V3

The following figure illustrates the internal task structure of the Sercos slave stack V3:

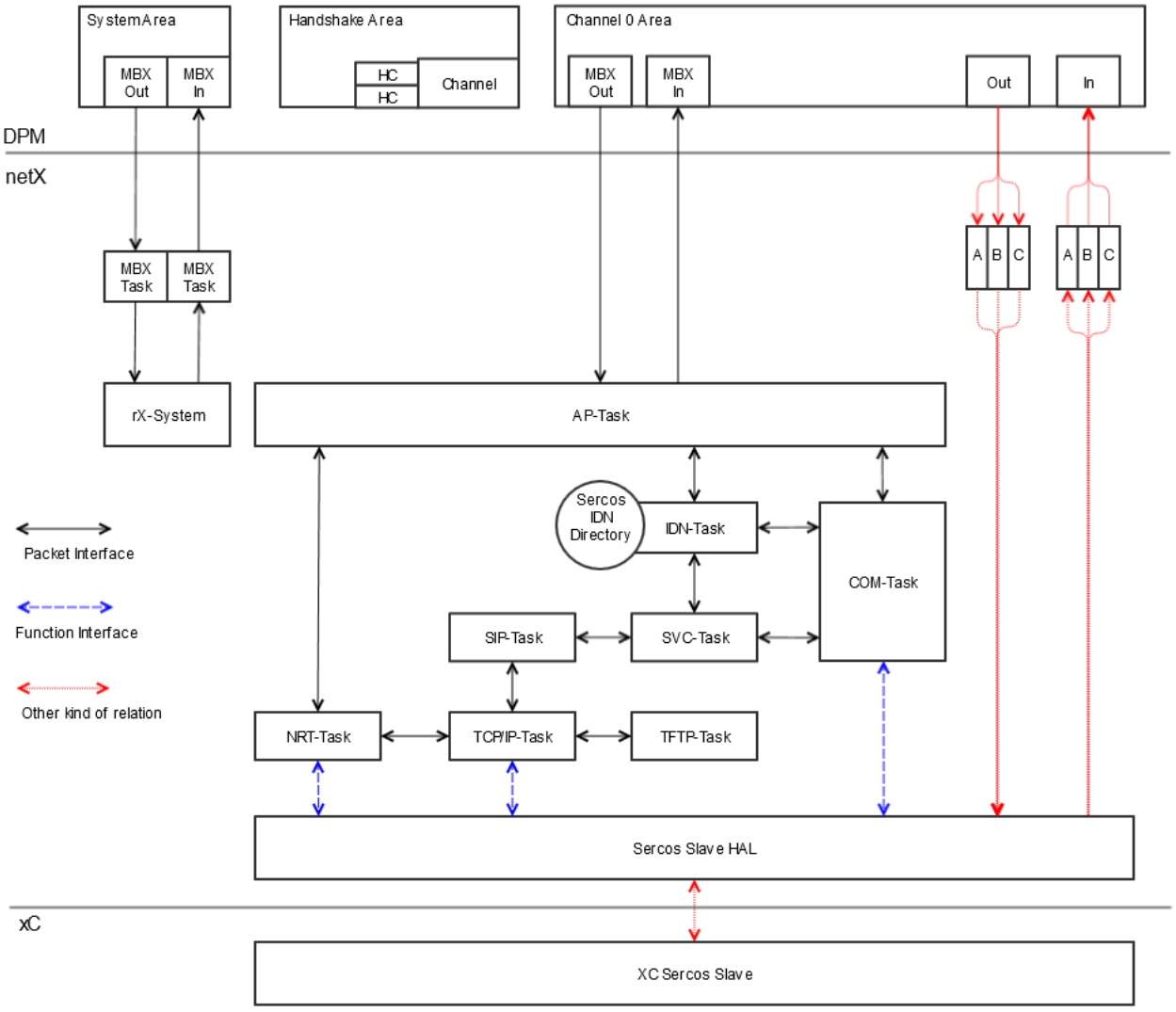


Figure 1: Task Structure of the Sercos Slave Stack

AP-Task

The AP-Task handles

- the dual-port memory
- the Sercos LED

COM-Task

The COM-Task handles

- the communication phases (state machine)
- cyclic data
- real-time bits

SVC-Task

The SVC-Task handles

- the service channel for read, write and commands

IDN-Task

The IDN-Task handles

- the IDN administration

NRT-Task

The NRT-Task handles

- sending and receiving of UCC frames (non-real-time Ethernet frames)
- It implements the Sercos part SCP_NRTPC.

SIP-Task

The SIP-Task handles

- SVC data transmission through UCC channel

2.2 Process Data (Input and Output)

The input and output data area is divided into the following sections:

I/O Offset	Area	Length (Byte)	Type
0x1000	Output block	5760	Write
0x2680	Input block	5760	Read

Table 6: Input and Output Data

The process data is limited based on the used netX chip. See section Technical Data on page 8 for more information.

2.3 Configuration Using the Packet API

This section explains how to configure the Sercos Protocol Stack by packet. In order to determine what has to be implemented and where to send the packets to, select one of the following scenarios the Sercos Protocol Stack can be run with.

Scenario: Loadable Firmware (LFW)

The host application and the Sercos Protocol Stack run on different processors. While the host application runs on a separate CPU, the Sercos Protocol Stack runs on the netX processor together with a connecting software layer, the AP task. The connection of host application and Protocol Stack is accomplished via a driver (Hilscher cifX Driver, Hilscher netX Driver) as software layer on the host side and the AP task as software layer on the netX side. Both communicate via a dual port memory.

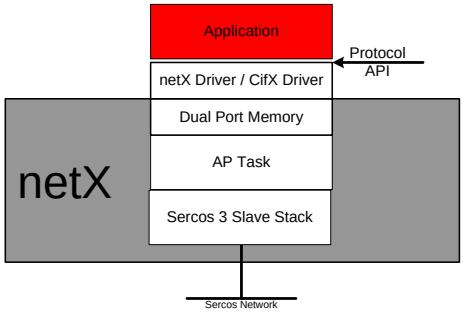


Figure 2: LFW

Scenario: Linkable Object Module (LOM)

Both the host application and the Sercos Protocol Stack run on the same processor, the netX. There is no need for drivers or a stack-specific AP task. Application and Protocol Stack are statically linked.

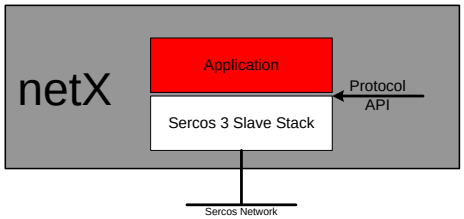


Figure 3: LOM

2.3.1 Stack configuration for Loadable Firmware

Required	Packet Name	Command Code	Page
Optional	SIII_SL_IDN_CMD_CREATE_IDN_REQ	0x5A40	78
Optional	SIII_SL_IDN_CMD_REGISTER_IDN_NOTIFY_REQ	0x5A44	82
Mandatory	SIII_SL_IDN_CMD_SET_CONFIGURATION_REQ	0x3628	60
Optional	SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_REQ	0x32B8	72
Mandatory	RCX_CHANNEL_INIT_REQ	0x2F80	See [1]
Optional	RCX_REGISTER_APP_REQ	0x2F10	See [1]
Optional	SIII_SL_IDN_CMD_CMD_REGISTER_CPX_CHECK_REQ	0x3240	135
Optional	SIII_SL_COM_CMD_INIT_COMPLETED_IND	0x3270	184
Optional	RCX_START_STOP_COMM_REQ	0x2F30	See [1]

Table 7: Packets for stack configuration for Loadable Firmware (LFW)

Within the following figure, the AP-Task is displayed. The packets are also routed to the Stack. So in all further figures explaining the packets e.g., the AP-Task will not be displayed any more also if you are using the loadable Firmware.

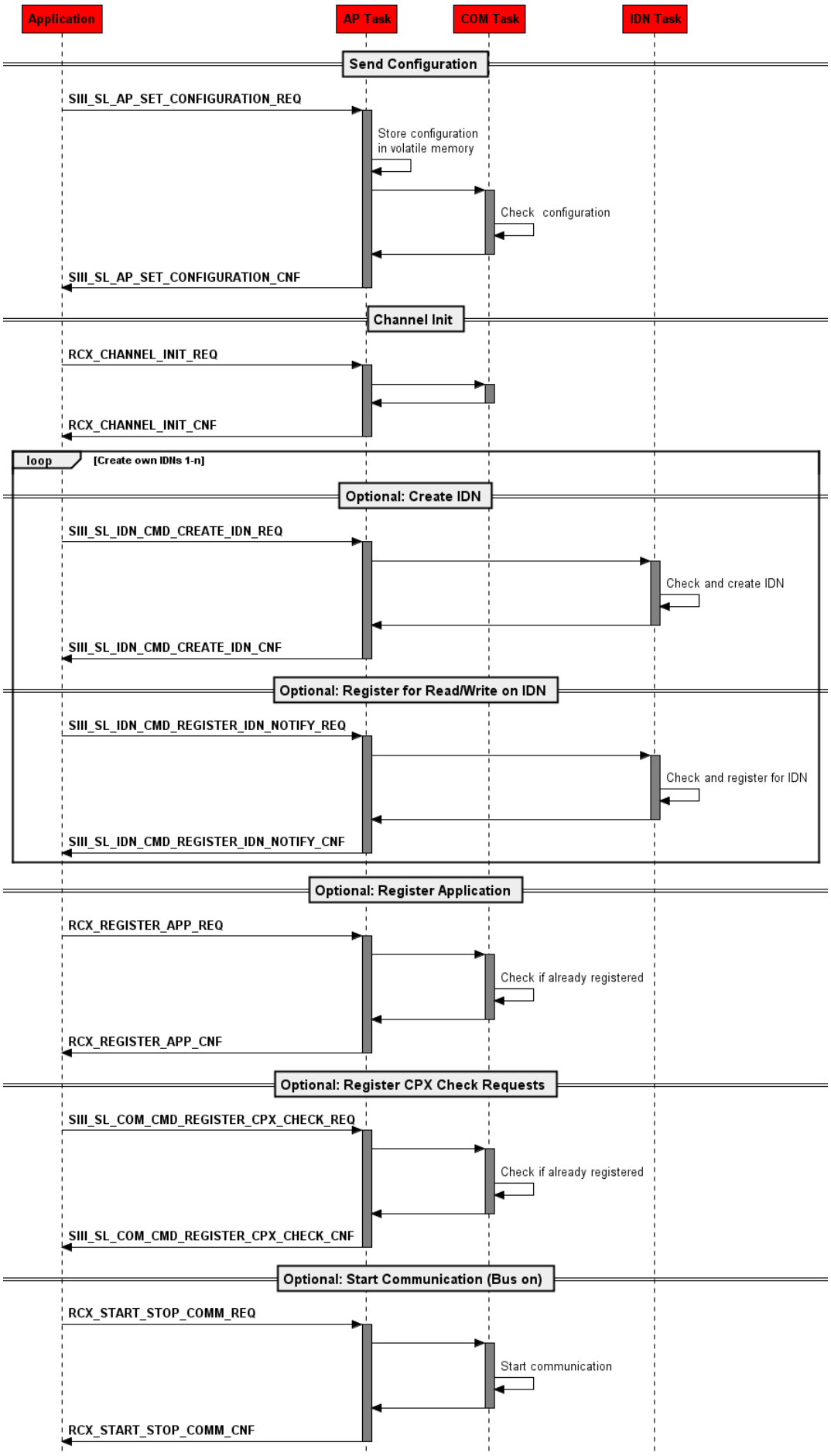


Figure 4: Startup sequence LFW

2.3.2 Stack configuration for Linkable Object Modules

Required	Packet Name	Command Code	Page
Optional	SIII_SL_IDN_CMD_CREATE_IDN_REQ	0x5A40	78
Optional	SIII_SL_IDN_CMD_REGISTER_IDN_NOTIFY_REQ	0x5A44	82
Mandatory	SIII_SL_IDN_CMD_SET_CONFIGURATION_REQ	0x3628	60
Mandatory	Register Callbacks S3Slave_SetupBufferFunctions()		59
Mandatory	RCX_CHANNEL_INIT_REQ	0x2F80	See [1]
Optional	RCX_REGISTER_APP_REQ	0x2F10	See [1]
Optional	SIII_SL_IDN_CMD_CMD_REGISTER_CPX_CHECK_REQ	0x3240	135
Optional	SIII_SL_COM_CMD_INIT_COMPLETED_IND	0x3270	184

If using the Linkable object module, all packets must be routed to the specific Stack task (e.g. Com-Task, IDN-Task).

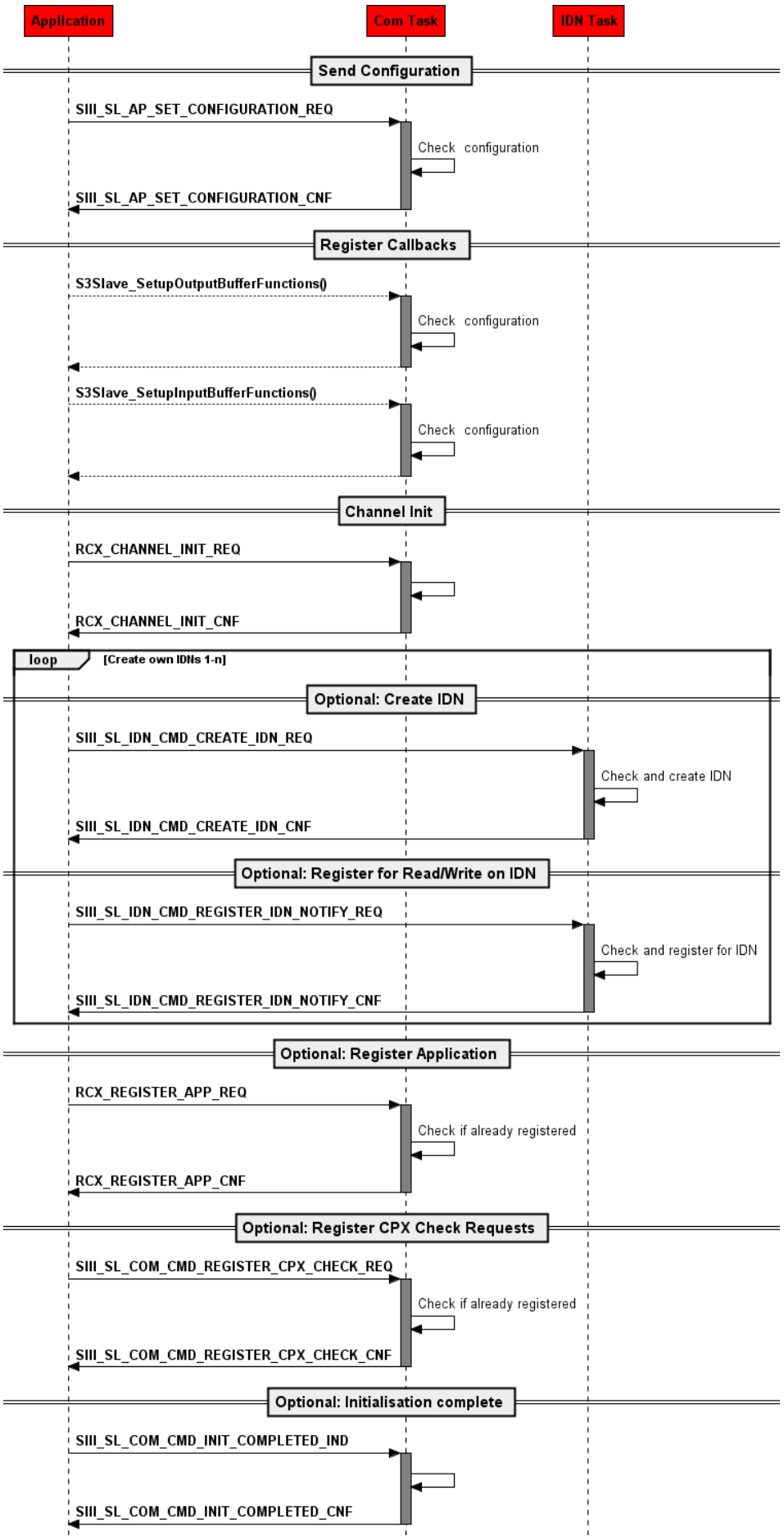


Figure 5: Startup sequence LOM

2.4 Configuration of UC Channel

The following information applies for the configuration of the UC Channel (Unified Communication Channel) of Sercos:

- In order to deactivate the UC channel, set the timing values t_6 to zero. This must be configured by the Sercos Master through IDN S-0-1017 (UC transmission time), see the documentation of the master for more information.
- The handling of Ethernet frames, especially the infiltration of the Ethernet frames into the SERCOS communication at the free port and the separation of Ethernet frames from the SERCOS communication are automatically done without any deterioration of the relevant timing parameters.

2.4.1 Loadable Firmware

All loadable firmware targets contain the NRT task. If this task is started NRT is possible and will be reported in IDN S-0-1000 (SCP_NRTPC will be reported). Nothing must be done, the flag `bSCP_NRT_Version` within Set-Configuration is don't care (see Chapter 5.3.1).

2.4.2 Linkable Object Module

The NRT task has to be started within the static task list at the `config.c` file. If the task is started the slave will support NRT and report SCP_NRTPC within IDN S-0-1000. If the NRT task is started the flag `bSCP_NRT_Version` within Set-Configuration does not care (see Chapter 5.3.1).

If the task is not started, but the flag `bSCP_NRT_Version` is set within the Set-Configuration packet, the stack will be forced to report SCP_NRTPC within IDN S-0-1000 but without the basic NRT-Task running.

2.5 Configuration of S/IP

The following information applies for the configuration of the S/IP (Sercos/IP Service) of Sercos:

2.5.1 Loadable Firmware

All loadable firmware targets contain the S/IP task. If the S/IP task is running the stack will report this in IDN S-0-1000 SCP_SIP. Nothing must be done, the flag `bSCP_SIP_Version` within Set-Configuration does not care (see section *Configure the Slave* on page 60).

2.5.2 Linkable Object Module

The SIP task and TCPIP task must started within the static task list at the `config.c` file to support SCP_SIP. If the SIP task is running the slave will support SIP and report SCP_SIP within IDN S-0-1000. If the SIP task is running the flag `bSCP_SIP_Version` within Set-Configuration does not care (see section *Configure the Slave* on page 60).

If the task is not started/running, but the flag `bSCP_SIP_Version` is set within Set-Configuration packet, the stack will be forced to report SCP_SIP within S-0-1000 but without the basic SIP-Task running.

3 General Topics

3.1 Start-up of the Sercos Slave Stack

After power on or reset the device performs a start-up initialization. During this phase several steps are taken to bring the device from the uninitialized state into operation.

After the start-up initialization the stack waits for the configuration parameters provided by the set-configuration packet followed by a channel init.

The Sercos device starts after initialization in the non-real-time mode (NRT). The real-time mode starts automatically after the first MDT0 telegram with communication phase 0 (CP0) was received from the master. If there are no more real-time telegrams in CP0, the device returns to NRT Mode and waits for new telegrams.

When phase CP4 is reached, the communication bit in the dual-port-memory is set and the host can exchange cyclic data.

Stack versions >V3.1.30.0 also support SCP_HP. The device could also startup over the hot plug phases if the Sercos bus is in CP4. This startup is possible if the master supports hot plug and has hot plug enabled in the MST0 CP4 frames. In this case the slave may startup over the phases HP0, HP1, HP2 to CP4. The phase HP2 equals to the phase CP3 and CP4.

At any time the host can access the data in the service channel object dictionary by using the packets `SIII_SL_IDN_CMD_READ_REQ` and `SIII_SL_IDN_CMD_WRITE_REQ`. By this access it is possible to set parameters e.g. the Electronic Label (similar to Manufacturer Version) IDN S-0-1300 or Minimum Feedback Processing Time S-0-1005 can be written before the Communication-Phase Startup occurs.

3.2 Communication Phases

The states which the Sercos master (and also the slave) can have during operation are called communication phases. There are communication phases numbered from 0 to 4 and, additionally, the non-real-time mode. This section discusses some of the most important aspects of the various communication phases and their transitions. It also explains the structure of the sent data telegrams depending on the communication phase.

3.2.1 State Transitions

The following illustration explains the possible transition between the states of the Sercos master:

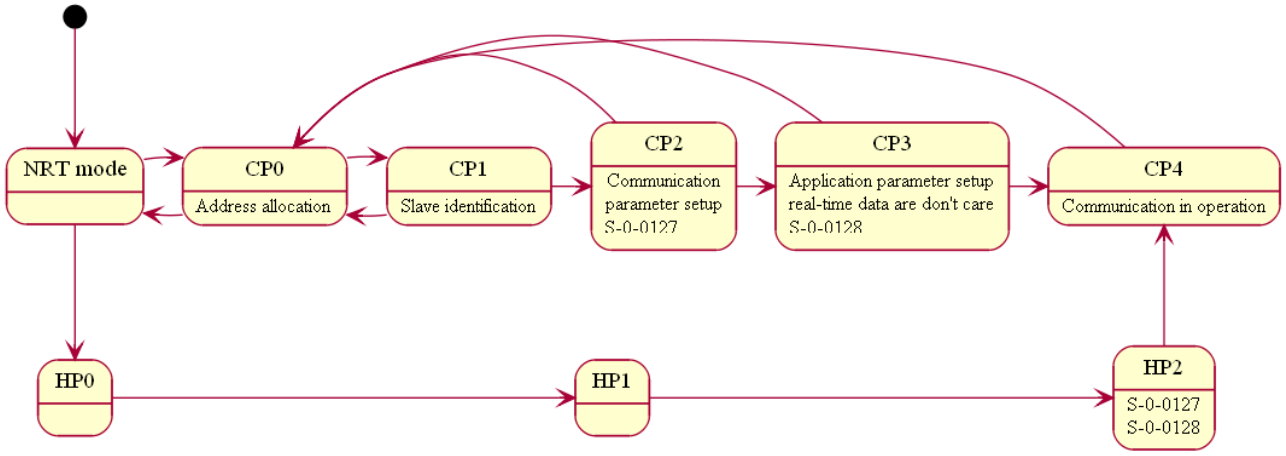


Figure 6: Permitted Communication Phase Transitions

When switching between two states the Sercos master is temporarily in an intermediate state CPS (Communication phase switching).

Note: Through MST losses or if a timeout occurs the device may switch back to NRT state. Before entering NRT state from a state different to CP0 or HP0 the slave generates a link down on both ports (behavior according specification). It can take up to 3 seconds until the link is up again. The master must be configured to wait long enough in CP0. If the CP0 timeout of the master is too low there is not enough time for the Slaves to get back in communication and insert data in the CP0 frames.

3.2.2 Non Real-Time Mode (NRT Mode)

In this mode, the Sercos device can operate as usual Ethernet node according to the Ethernet standard IEEE 802.3, i.e. standard Ethernet frames are handled and re-sent. This mode also serves as a power-up mode meaning after power has been switched on and the internal checks of the SERCOS slave have successfully been finished, the SERCOS slave will reach this state if a set-configuration followed by a channel init succeeded.

Transition to Communication Phase 0

One of the most important information items the Sercos Master requires is the topology of the network. Therefore it starts topology recognition already in this early stage when trying to switch from NRT mode to CP0. In order to do so, it sends 10 MDT0 telegrams both on the primary and the secondary channel and depending on the received answer telegrams it decides which kind of network has been detected if any has been detected at all. The following results are possible:

- Line topology, only primary channel used
- Line topology, only secondary channel used
- Double line topology, both channels used
- Ring topology, both channels used

Applied cycle times range from 1 to 65 ms. If the topology recognition was successful, the Sercos Master should switch to CP0.

3.2.3 Communication Phase 0

Then the Sercos Master internally stores the result of preceding successful topology recognition which is available in CP0 up to CP4.

In CP0, the Sercos Master performs a measurement of the ring delay time which is needed for internal timing calculations as several timing-related settings of the Sercos communication network which need to be adjusted within CP0 depend on this specific value. The Sercos master also sends MDT0 telegrams to the Sercos slave must respond with the corresponding AT0 telegram. This gives the Sercos master the opportunity to check the presence of slaves in the network and to collect first slave-related information.

CP0 is the only state which can be entered from any other state as it can be entered both from any higher communication phase and from NRT mode such as the topology address.

The service channel for acyclic communication is not active in CP0. If the Sercos slave is in CP0 and does not receive an MDT0 telegram from the Sercos master it will fall back to NRT mode.

Transition to Communication Phase 1

The Sercos Master sends telegrams with identical content to the slaves expecting identical answers. According to specification, the Sercos Master is ready to switch into the next phase if the received AT telegrams contain identical data for N cycles. N could be configured and has usually the value 100.

3.2.4 Communication Phase 1

There are two necessary conditions under which the Sercos network will proceed from CP0 to CP1:

- The Sercos master has received N (usually 100) AT telegrams with identical contents.
- The Sercos master will not proceed if the bus consists of two separated line. The bus must be changed to a single line or a ring.
- If the Sercos master has a configuration loaded, it will only proceed, if the expected and found slave addresses match.

Then the master will switch to CP1 on its own initiative.

The main purpose of CP1 is the identification of slaves. During CP1 the Sercos master checks the presence of all configured Sercos devices and performs tests whether the Sercos slaves should be activated for the higher communication phases or not. Depending on the configured number of slaves the Master sends two or four MDT respectively two or four AT telegrams. This is accomplished by the Sercos master by sending the telegrams MDT0 and MDT1. The Sercos master then expects an answer from the Sercos slaves within telegrams AT0 and AT1. If necessary, MDT2 and MDT3 and AT2 and AT3 are used additionally. The position of the SVC control and status word corresponds with the physical position of the slave (TADR).

The service channel for acyclic communication is initialized in CP1 as it is used for the identification of slaves. The UC channel is active (optional).

Transition to Communication Phase 2

Switching to phase 2 can start if SVC control bit MHS(1) has been confirmed by AHS=1, SVC valid=1(SVC status) and slave valid=1 (Device status).

3.2.5 Communication Phase 2

After having decided which slaves to accept for higher communication phases the Sercos master will switch to CP2 on its own initiative. The correctness of placing the control and status words within the AT is checked and the decision whether to switch to CP2 or not is mainly made depending on the result of this check.

The main purpose of CP2 is to transmit the most important communication parameters from the Sercos Master to the slaves via the service channel. Parameters for cyclic communication and timing of messages which have been calculated in CP2 or prior or which have been loaded from the configuration database are set now and some other initializations are performed. Especially, in CP2 the Sercos Master has to write this value to IDN S-0-1015 “Ring Delay” of every slave.

Note: Slaves which were not activated in CP1 must not react in CP2 (or even higher) but issue an error message.

The service channel for acyclic communication shall be completely active at the slave in CP2.

Transition to Communication Phase 3

When having done all preparations necessary in CP2, the Sercos master will on its own send a procedure command (IDN S-0-127) to every accepted Sercos slave in order to test the readiness of these slaves and to prepare switching to CP3. If all Sercos slaves successfully perform this procedure command, the Sercos master will subsequently switch to CP3.

If the transition to Communication phase 3 fails

If the transition fails, the slave will place the source of the error in S-0-0021. The IDNs reported here a configured false. Change the content of this IDN and retry a phase transition to CP3.

3.2.6 Communication Phase 3

If all Sercos slaves successfully perform the procedure command (IDN S-0-127), the Sercos Master switches to CP3.

Communication Phase 3 serves as a kind of test state in which real data communication between master and slave using the mechanisms of CP4 is performed without offering the full functionality of CP4. Especially the telegrams exchanged between master and slave do not differ at all between CP3 and CP4, the master sends the exactly configured MDTs and ATs to all slaves.

All parameters which have not yet been configured in the earlier states are configured now. All IDNs for the slaves signed by CP3 transition flag are set using the service channel.

Device control bits and device status bits are activated in CP3. The process data is not valid now.

The service channel for acyclic communication is completely active at the slave in CP3. Transmission reliability for the service channel shall be guaranteed by the MHS and AHS-bits as well as the HS timeout.

Also the IP channel is fully active in CP3.

Note: The IP channel can be deactivated by setting both t6 and t7 to the same value.

Transition to Communication Phase 4

When having done all preparations necessary in CP3, the Sercos master will on its own send a procedure command (IDN S-0-128) to every accepted Sercos slave in order to synchronize master and slaves and to prepare switching to CP4. If all Sercos slaves successfully perform this procedure command, the Sercos Master will subsequently switch to CP4. During processing of this procedure command the slave has to check the validity of the transmitted parameters for CP4 and to activate the synchronization.

If the transition to Communication phase 4 fails

If the transition fails, the slave will place the source of the error in S-0-0022. The IDNs reported here a configured false. Change the content of this IDN and retry a phase transition to CP4.

3.2.7 Communication Phase 4

If all Sercos slaves successfully perform the procedure command (IDN S-0-128), the Sercos master switches to CP4. Communication phase 4 offers the full functionality of the Sercos network including service channel and IP channel. The process data is valid now.

For more information about the details of the Sercos communication phases please refer to the Sercos specification.

3.2.8 Hot plug

The hot plug phases 0-1 are needed to configure the slave to take part to the bus communication. The phase HP2 equals to the phase CP2. In this phase everything is done as in CP2. The only difference is there is no CP3. After the S-0-0127 procedure command executed the following behavior is according CP3 but the slave stays in HP2. Past the execution of the S-0-0128 procedure command the slave switches directly to CP4.

3.3 Data Exchange

3.3.1 Cyclic Data Exchange

The cyclic data exchange is used to transport real-time data from the bus to the host and vice versa. The cyclic data (see *netX DPM Interface Manual - Section "Input/Output Data Image"*) is available after Communication Phase 4 is reached. The RT-Valid Bit is always set by the stack. The layout of the process data is defined in the packet `SIII_SL_AP_SET_CONFIGURATION_REQ_T`. E.g. it is possible to locate the connection control at the offset 0 in input and output data. The length of the connection control is always 2 bytes. The process data (input or output, depending on the direction) can be located at offset 2, just behind the Connection Control. This entry contains connection-specific data.

For more information, see section *Connection Control (C-Con)* on page 40.

3.3.1.1 Bus Synchronous Mode

The bus synchronous mode is the common method for data exchange. This mode is enabled by setting `bProcDataMode` to `SIII_SL_AP_SET_CONFIGURATION_DEVICE_PROCDATA_MODE_BUS_SYNC`. This data exchange mode will always have a stable round-trip time of 3 Sercos cycles. If the application has received the data from the MDT block, the application has almost a sercos cycle to provide the answer in the third cycle.

This mode is independently of the bus configuration. This also applies for MDT/AT/UCC or MDT/UCC/AT configurations.

The diagrams below explain the sequence of actions leading to a round trip time of 3 cycles both for the LFW and the LOM scenario.

LFW scenario:

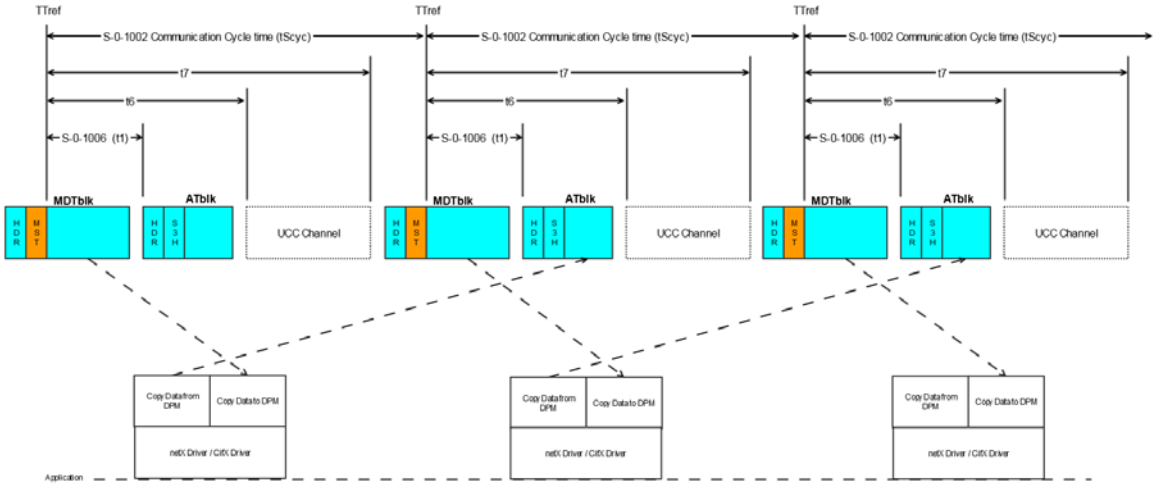


Figure 7: Bus Synchronous mode (LFW scenario)

LOM scenario:

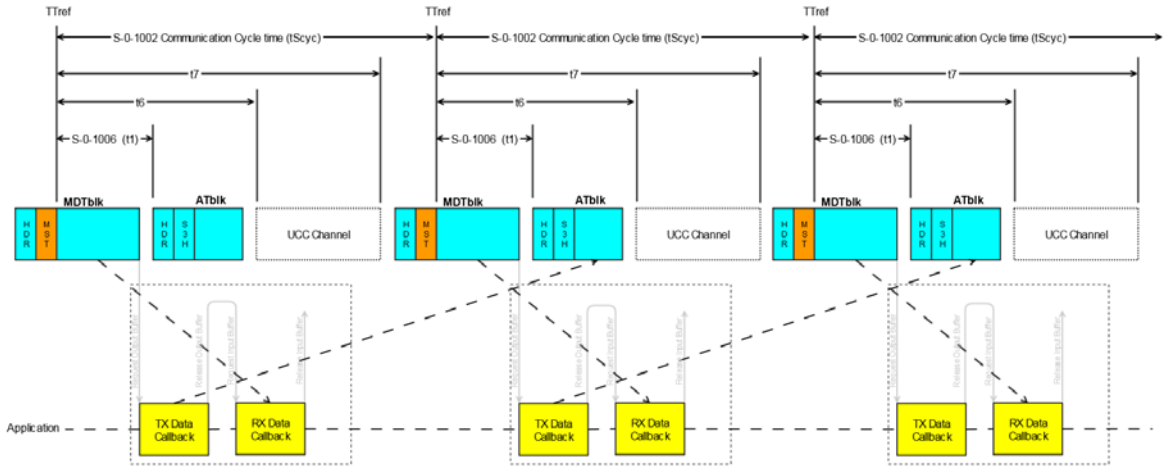


Figure 8: Bus Synchronous mode (LOM scenario)

3.3.1.2 Bus Synchronous Mode with reduced round-trip time

The bus synchronous mode with a reduced round-trip time is enabled by setting `bProcDataMode` to `SIII_SL_AP_SET_CONFIGURATION_DEVICE_PROCDATA_MODE_SEPARATE_AT_CALLBACK`. For more information on this setting see *Table 20* on page 64.

With this buffer mode it is possible to provide an answer on the received data within just a single Sercos cycle. However, the following restrictions apply:

- This mode is not possible for small cycle times and also dependent on the length of the IO data, the application processing time and the configuration of the bus-timings made by the Sercos Master.
- This mode is only possible for a MDT/UCC/AT bus configuration, because MDT/AT/UCC will cause a delay of at least one Sercos cycle.

For this mode the stack calculates the minimum feedback processing time t_5 (if necessary it is possible to adjust t_5 by application within IDN S-0-0127). t_5 depends on the length of the IO-Data the stack needs for processing the data.

Adapting t_5 can require effort, e.g. an ETM interface on the target device can be needed.

LFW scenario:

The IO data must be present in the DPM at the beginning of t_5.

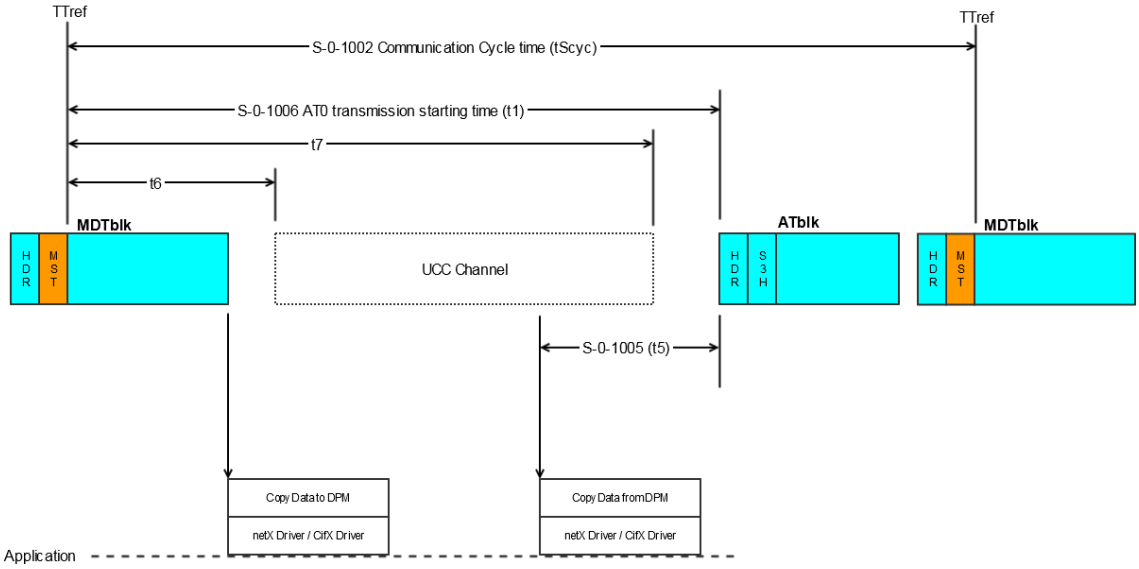


Figure 9: Bus Synchronous mode with reduced round-trip time (LFW scenario)

LOM scenario:

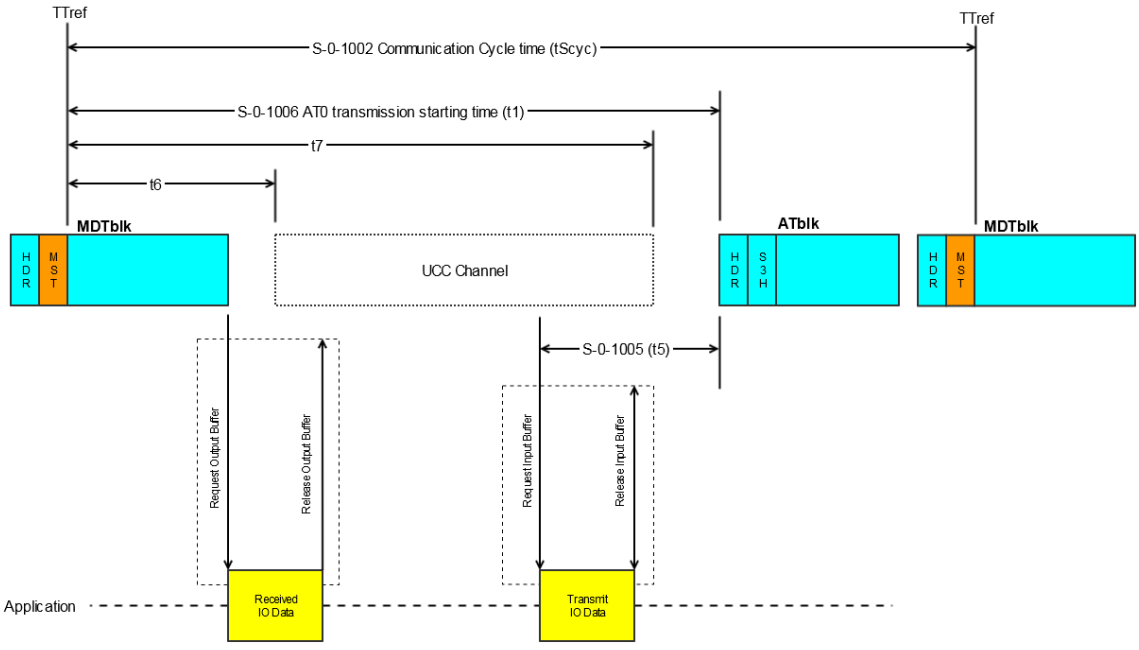


Figure 10: Bus Synchronous mode with reduced round-trip time (LOM scenario)

3.3.1.3 Connections for Real-Time Data

The stack uses descriptors for the connection configuration. Totally, there are 64 descriptors available for the connection configuration. Also the Service Channel (SVC) and Connection Control (CCon) need one descriptor per data direction. So for each slave this lowers the amount of available descriptors by 4. Configuration of the SVC and CCon on descriptors is done by the stack.

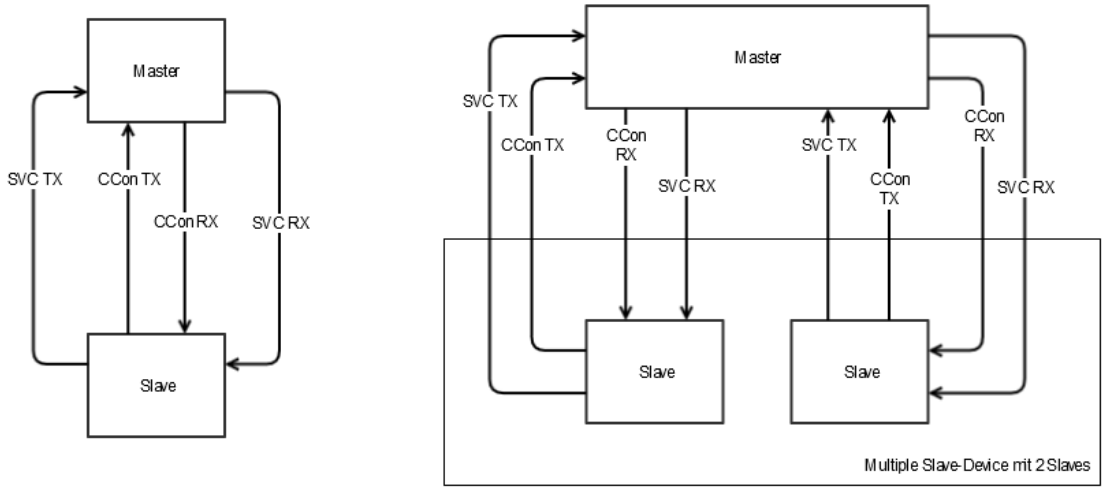


Figure 11: Master-slave communication for slave devices with one or two slaves

The following table lists the number of available IO Connections in dependence of the number of slaves in the device.

Slaves in Device	Available for IO connections
1	60
2	56
3	52
4	48
5	44
6	40
7	36
8	32

Figure 12: Number of available IO connections in dependence of the number of slaves

Fix Configuration (FixCfg)

A device with Fix Configuration has per definition one Consumer and one Producer Connection. The connection producing data may be placed in any AT. The connection consuming data may either be placed in any MDT or AT.

Var Configuration (VarCfg)

A certain number of connections are supported. The slave defines this number of connections and provides it to the master. Each consuming connection is configurable in every MDT or AT. Each producing connection is configurable in every AT.

The slave provides lists of IDNs, which contains the IDN of produced or consumed data. The content of the connections has to be configured by the master based on the lists provided by the slave.

For more information on FixCfg and VarCfg see section *SCP FixCFG or VarCFG* on page 41.

Cross Communication (CC)

Cross Communication is possible both with a FixCfg and VarCfg Slave. If a slave consumes data from an AT this is known as Cross Communication. The received AT data are first available after the following MDT block.

3.3.2 Acyclic Data Exchange through IDNs

The objects within the object dictionary, which are uniquely identified by their IDNs, represent the entire parameter set of the Sercos slave.

This section only concentrates on the most important IDNs, which represent the main parameters of the Sercos slave and will often be accessed.

You might wish to implement a lot of these IDNs in your application as the application shall be ready to react to these, so the aim of this chapter is to describe the general access procedure and to explain some of the most important commonly used IDNs.

3.3.2.1 General Access Procedure

The object dictionary is accessed via the Sercos service channel or through SIP. The single objects are represented by their unique IDN. Assigned to each IDN is a data block consisting of 7 elements with the following structure:

Element	Name	Type	Mandatory/Optional
1	IDN structure	32-bit value	Mandatory
2	Name (of operational data)	Text, UTF8 (Max. 244 bytes wide), max. 60 visible characters	Optional
3	Attribute (of operational data)	Bit mask (4 bytes wide)	Mandatory
4	Unit (of operational data)	Text, UTF8 (Max. 52 bytes wide), max. 12 visible characters	Optional
5	Minimum allowed input value	As #7 (Max. 8 bytes wide)	Optional
6	Maximum allowed input value	As #7 (Max. 8 bytes wide)	Optional
7	Data	4 different alternatives	Mandatory

Table 8: Available Elements of an IDN

3.3.2.2 IDN Structure

As already stated above, each IDN is uniquely assigned to an object which can be seen as a block of data containing slave-related data. In general, Sercos offers a space of 2^{32} allowed IDNs.

An IDN is displayed as `<S/P>-<PS>-<Data Block Number>.<SI>.<SE>` e.g. `S-0-1040.0.0`, `P-0-1000.0.0` or `S-0-1303.0.10`

Bit No.	Description
31-24	Structure Instance (“SI”)
23-16	Structure Element (“SE”)
15	S/P Parameter (“S/P”)
14-12	Parameter Set (“PS”)
11-0	Data Block Number

Table 9: Structure of an IDN

The following rules divide this space into subspaces:

- Subdivision between standard data (S) and product-related (or manufacturer-specific) data (P). This is accomplished by setting bit 15 of the IDN:

  0 - for standard data,

  1 - for product-related data.

- Subdivision between 8 separate parameter sets (denominated as parameter sets 0 to 7). Each of these parameter sets may then contain 4096 (equivalent to 2^{12}) IDNs.
- SE: 0-127, are used for Standard parameters
- SI: 0-255, addressing the structure of the same type not defined by Sercos

The IDNs are generally transferred as 32-bit values within the Sercos telegrams

3.3.2.3 Name

This optional element holds the name of the operational data which are stored under the respective IDN.

The length is limited to at most 244 bytes. At least 4 bytes need to be used.

These bytes are structured as follows:

- The first two bytes contain the hexadecimally coded value of the length of the programmed text. This is the text the master proposes to the slave. If these two bytes are 0, a zero-length name will be defined therefore.
- The next two bytes contain the hexadecimally coded value of the maximum allowed length of this text if the slave is permitted to change the text. (If this length is equal to 0, the slave is not permitted to do so.)
- Beginning from the fifth byte there is a string consisting of up to 240 bytes (UTF8, up to 60 visible characters) space for the actual name of the object assigned to the IDN. Characters exceeding the amount specified in the length bytes should be truncated by the Sercos slaves.

For more information, refer to the Sercos specification.

3.3.2.4 Attribute

This mandatory element contains additional information required for administrative purposes.

It is a 32-bit wide bit mask to be interpreted according to the subsequent table:

Bit No.	Description
31	Reserved
30	Write protection in CP4: 0 - Write protection not effective for operation data 1 - Write protection effective for operation data
29	Write protection in CP3: 0 - Write protection not effective for operation data 1 - Write protection effective for operation data
28	Write protection in CP2: 0 - Write protection not effective for operation data 1 - Write protection effective for operation data
27-24	Position of decimal point for input and display (not applicable to floating point data) 0000 - No places following the decimal point … 1111 - 15 places following the decimal point
23	Reserved
22-20	Coding for data type and display format: Data type [Display format] 000 - Binary value [Binary] 001 - Unsigned integer [Decimal] 010 - Signed integer [Decimal + sign] 011 - Unsigned integer [Hexadecimal] 100 - Extended character set [Text] 101 - Unsigned integer [IDN] 110 - ANSI 754-1985 floating point number (single precision) [Decimal value with exponent (fraction after decimal point is not taken into account)] 111 - SERCOS time[Display format: according to IEC 61588 4 octets seconds & 4 octets nanoseconds, starts with 1.1.1970 computed in UTC]
19	Function: 0 - Operation data/parameter 1 - Command
18-16	Data length (required for correct termination of data transmission on the service channel): 000 – Reserved 001 - Two bytes of operation data 010 - Four bytes of operation data 011 - Eight bytes of operation data 100 - Length is variable/1-byte data strings 101 - Length is variable/2-byte data strings 110 - Length is variable/4-byte data strings 111 - Length is variable/8-byte data strings
15-0	Conversion factor used for conversion of data to display format, specified as unsigned integer. Use 1 if not required (for instance for binary, character string or floating point number data)

Table 10: Coding of Attribute Information in IDN

The display format and the data length must match. Corresponding combinations are marked in the table below:

For more information on the extended character set see the specification, appendix E.

3.3.2.5 Unit

This optional element holds the name of unit to be applied to the operational data which are stored under the respective IDN.

The length is limited to at most 52 bytes. At least 4 bytes need to be used.

These bytes are structured as follows:

- The first two bytes contain the hexadecimally coded value of the length of the programmed text. This is the text the master proposes to the slave. If these two bytes are 0, a zero-length name will be defined therefore.
- The next two bytes contain the hexadecimally coded value of the maximum allowed length of this text if the slave is permitted to change the text. (If this length is equal to 0, the slave is not permitted to do so.)
- Beginning from the fifth byte there is a string consisting of up to 48 bytes (UTF8, up to 12 visible characters) space for the actual unit of the object assigned to the IDN. Characters exceeding the amount specified in the length bytes should be truncated by the Sercos slaves.

When the data type is either binary or character string, the data has no unit. For more information, refer to the Sercos specification.

3.3.2.6 Minimum / Maximum

This optional element holds the minimum or maximum value allowed for the operational data which are stored under the respective IDN. Lower or higher values respective cannot be processed by the slave, i.e. when a write request occurs with a lower or higher value, the original value will not be changed.

This element is not applicable in the following cases:

- Working with binary numbers
- Working with character strings
- Operation data have variable length

Note: Maximum can also be used for binary data according to the Sercos standard, set bits are supported by the slave.

3.3.2.7 Data

There are 4 formats defined in the Sercos standard which can be applied here:

- Fixed length format with 2 bytes
- Fixed length format with 4 bytes
- Fixed length format with 8 bytes
- Variable length format with support for up theoretically up to 65532 bytes.

In case of the variable length format these bytes are structured as follows:

- The first two bytes contain the hexadecimally coded value of the current length of the data. This is the text the master proposes to the slave. If these two bytes are 0, a zero-length datum will be defined therefore.

- The next two bytes contain the hexadecimally coded value of the maximum allowed length of data if the slave is permitted to change the text. (If this length is equal to 0, the slave is not permitted to do so.)
- Beginning from the fifth byte there is a string consisting of up to 65532 bytes space for the data of the object assigned to the IDN. Characters exceeding the amount specified in the length bytes should be truncated by the Sercos slaves. Depending on the attribute of the IDN, the data type is a 1, 2, 4 or 8 byte list.

3.4 MDT Real Time Data

The MDT Frames are used to transmit data from the master to the slaves (except CC connections). The master copies the data into the offsets for every slave. Each slave gets two connection configurations S-0-1050.0.1 and S-0-1050.1.1. Within this configuration it is determined whether the connection is a producer or consumer connection. The offsets and sizes of the individual slave connection are configured by the master. C-Con and IO-Status can get handled by the slave stack or by the application. If the device is an FSP IO-Device the IO-Status has to be evaluated. For a FSP Drive device there will be no IO-State in the connection.

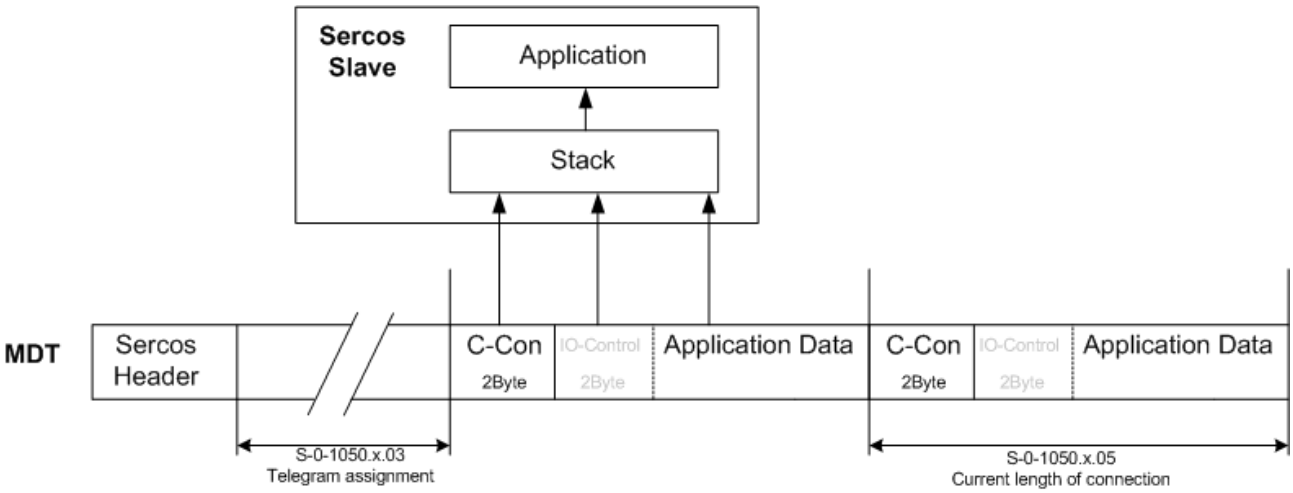


Figure 13: MDT Connection

3.5 AT Real Time Data

The AT Frames are used to transmit data from a slave to the master (except CC connections). The slave copies the data into the offsets at the passing frame. Each slave gets two connection configurations S-0-1050.0.1 and S-0-1050.1.1. Within this configuration it is determined whether the connection is a producer or consumer connection. The offsets and sizes of the individual slave connection are configured by the master. C-Con and IO-Status can get handled by the slave stack or by the application. If the device is an FSP IO-Device the IO-Status has to be served. For a FSP Drive device there will be no IO-State in the connection.

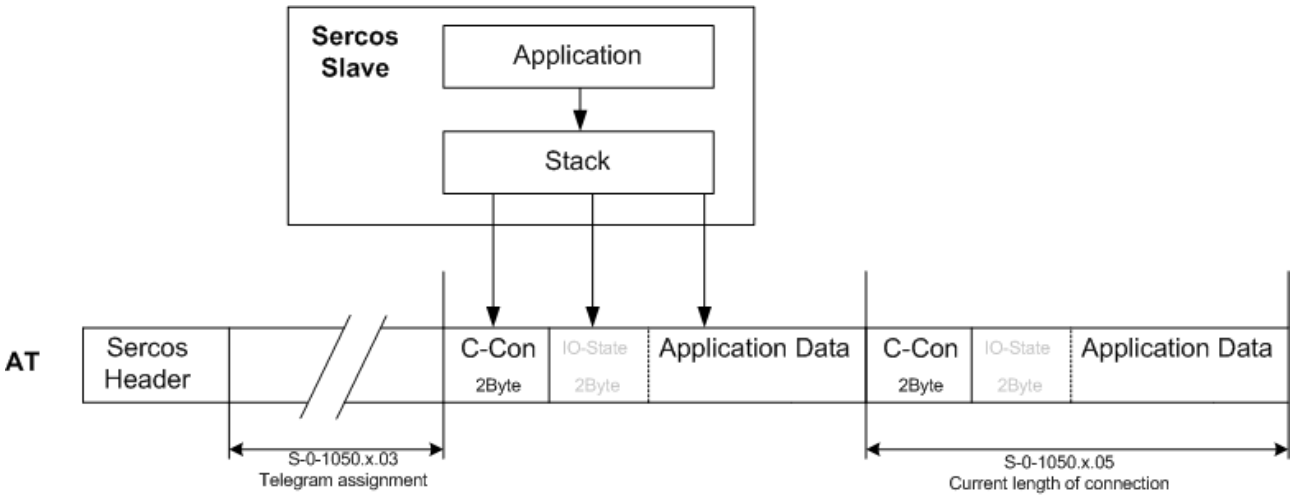


Figure 14: AT Connection

3.6 Connection Control (C-Con)

This item corresponds to IDN S-0-1050.x.8 Connection Control

Bit No.	Description
15-12	▪ initial value of this counter is 0 in CP4. ▪ every change of this counter announces new application data in the connection and the application data may be processed. ▪ at a clock synchronous connection this counter shall always be increased in the related Sercos cycle. ▪ at a non-synchronous connection the change of this counter triggers a watchdog, i.e. after change of this counter the monitoring time (*tPcyc*) is always started again. This counter shall be increased once per monitoring time (*tPcyc*), after producer ready (bit 0) is set to 1. ▪ Bit 12 shall be equal to bit 1 „*New data*“.
11-10	Reserved
9	Reserved for real-time bit 4
8	Reserved for real-time bit 3
7	Real-time bit 2
6	Real-time bit 1
5	Reserved
4	Flow-control 0 - run - producing and consuming are active. 1 – stop - The producing is canceled, the consumer shall not generate an error.
3	Reserved
2	Data field delay (consumer shall prefer taking the data of the port at which this bit has the value 0): 0 - No delay - Data are transmitted without delay in the same Sercos cycle. 1 – delay - Master has copied the data and therefore the data are transmitted with additional delay of one Sercos cycle.
1	New data (New producer data): Toggle - detailed description of new data bit: - initial value of this bit is 0 in CP4 - when changes occur, this bit has to be toggled - every toggle of this bit announces new data in the connection and then the data are exchanged between connection and application. This implies that in the consumer when the data of a certain producer cycle has not been received, the prospected value for this bit has also to be toggled for the next communication cycle. - at a cycle synchronous connection this bit shall always be toggled in the related Sercos cycle. - at an asynchronous connection this bit triggers a watchdog, i.e. after toggle of this bit the monitoring is always started again with the monitoring time (t_{Pcyc}). - this bit shall be toggled once per monitoring time (t_{Pcyc}), after producer ready (bit 0) is set to 1.
0	Producer ready 0 - not valid - The producer does not generate any data in this connection yet 1 – valid - The producer generates data in this connection. The consumer can process the connection data if the producer has toggled the New data (bit 1). The producer ready bit shall be evaluated in CP4 only.

Table 11: Connection Control IDN S-0-1050.x.8

3.7 SCP FixCFG or VarCFG

A device can be built as SCP FixCFG or SCP VarCFG device. The type `ulFSP_Type` is configured using the Set Configuration packet.

A Sercos Slave reports the SCP Type in IDN S-0-1000.

3.7.1 SCP FixCFG

An SCP FixCFG slave has exactly two connections, one as consumer and one as producer. The content of the connections is defined by the slave. The master cannot change the content of the connections.

SCP Type FixCFG will be configured by the Set Configuration packet (see section *Configuration Using the Packet API* on page 15).

3.7.2 SCP VarCFG

An SCP VarCFG slave supports a certain number of connections. The number of connections is defined by the slave. The content of all connections is configured by the master. The slave provides lists of IDNs for the master to determine which configuration is allowed.

The application has to configure the IDNs S-0-0187 and S-0-0188.

The configuration of SCP VarCFG is done by the Set Configuration packet. The IDNs S-0-0188 and S-0-0187 must contain the allowed connections the master may configure. The stack calculates the “Current length of the connection” (S-0-1050.x.05) based on the IDNs S-0-0188 and S-0-0187 dependent on the configuration made in S-0-1050.x.01 by the master.

Several configuration methods exist that a Sercos master can use. The Sercos master can use single parameters or a container to configure a slave.

As required by the Sercos specification, the Sercos slave stack automatically adds a padding byte after each odd connection length, if the master uses single parameters to configure the connection.

As required by the Sercos specification, the Sercos slave stack automatically adds a padding byte after an odd connection length of a container, if the master uses a container to configure the connection.

In order to get offsets of dual-port memory data which are independent from the configuration method of the master, we strongly recommend using even container lengths only.

3.7.2.1 Consumer Configuration

The following example shows how the master configures the IO-Status S-0-1500.0.2 and several FSP IO IDNs into the Connection.

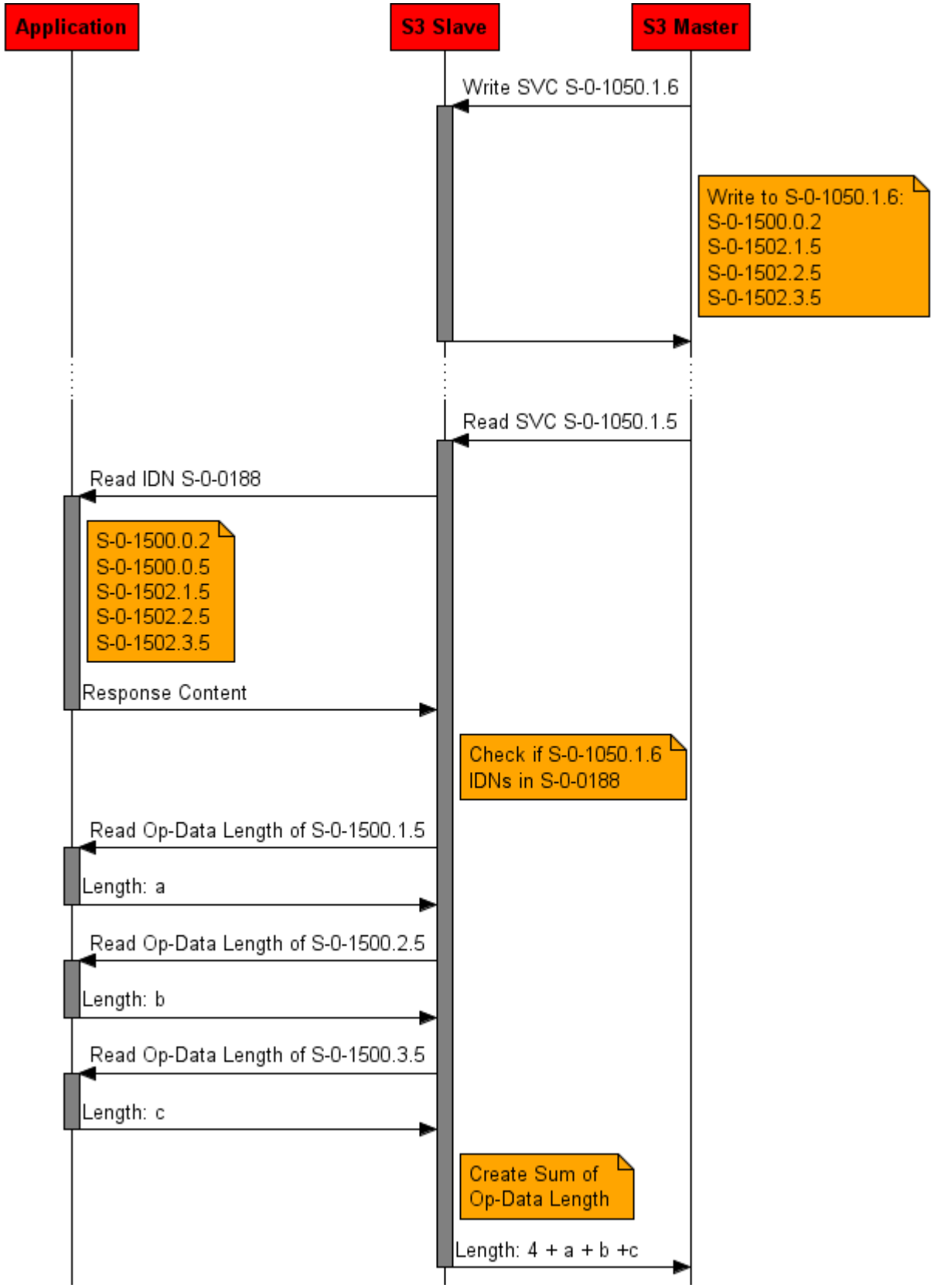


Figure 15: FSP VarCFG Consumer configured with single parameter

The following example shows how the master configures the IO-Status S-0-1500.0.2 and just the “Container Output Data” S-0-1500.0.5 into the Connection. The “Container Output Data” should also insist the connection IO FG IDNs.

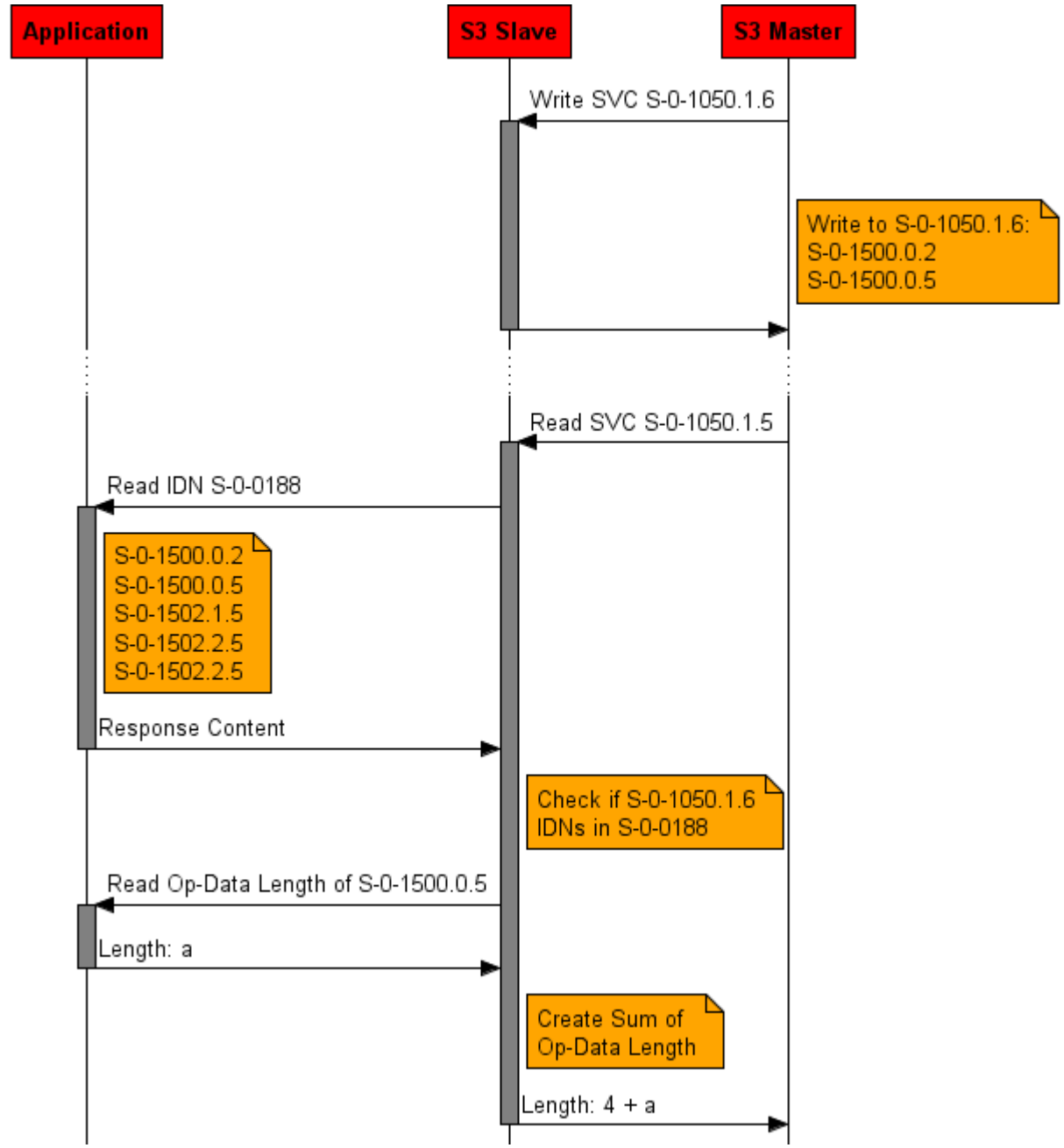


Figure 16: FSP VarCFG Consumer configured with container

3.7.2.2 Producer Configuration

The following example shows how the master configures the IO-Status S-0-1500.0.2 and several FSP IO IDNs into the Connection.

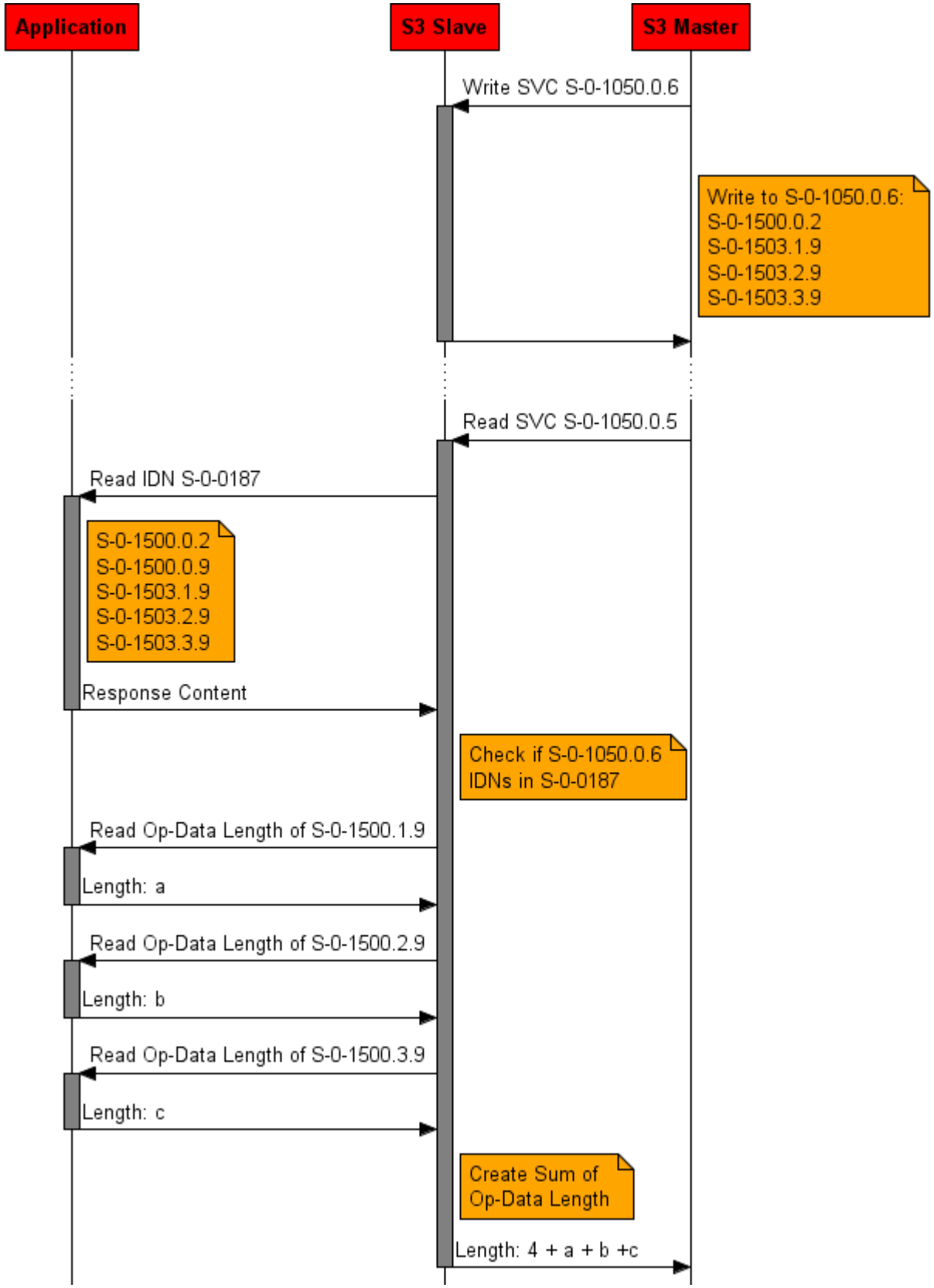


Figure 17: FSP VarCFG Producer configured with single parameter

The following example shows how the master configures IO-Status S-0-1500.0.2 and just the “Container Input Data” S-0-1500.0.9 into the Connection. The “Container Input Data” should also insist the connection IO FG IDNs.

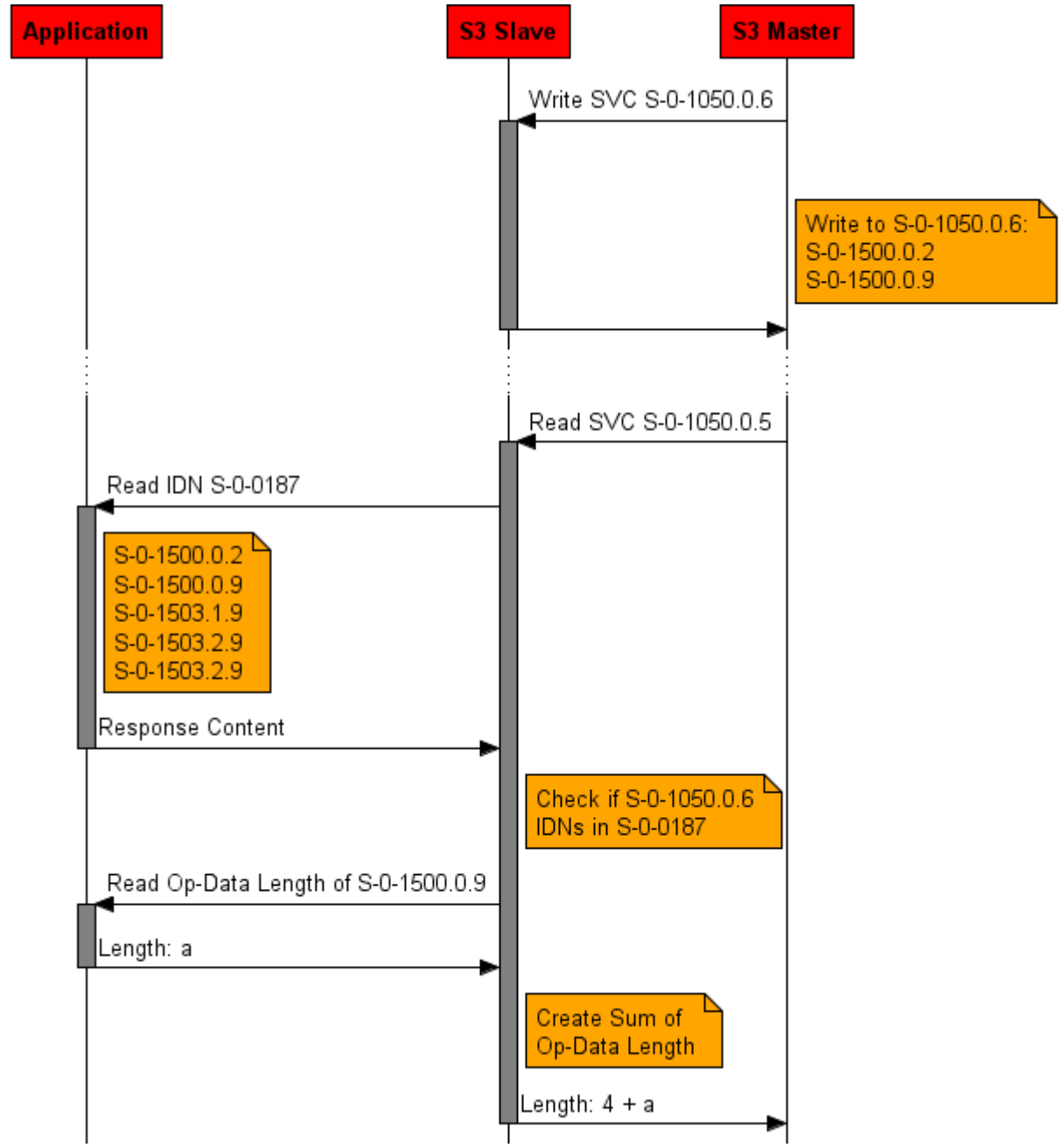


Figure 18: FSP VarCFG Producer configured with container

3.8 FSP Type and Version

The „FSP Type & Version“ S-0-1302.x.1 indicates the function specific type and version of the resource. At the moment there are three different possible types: “FSP IO”, “FSP Drive” and “FSP Encoder”. Dependent on the FSP Type a special handling is necessary. The handling and additional required IDNs are described in the Sercos specification. The following subsection will give a short introduction into FSP IO.

3.8.1 FSP IO

For FSP IO slaves the following IDNs are mandatory:

- S-0-1500.x.01 IO Control
- S-0-1500.x.02 IO Status
- S-0-1500.x.03 List of module type codes
- S-0-1500.x.05 Container Output Data
- S-0-1500.x.09 Container Input Data

3.8.1.1 IO-Status

This item corresponds to IDN S-0-1500.x.2 IO Status

Bit No.	Value	Description	Comments
15		outputs ready-to-operate	mandatory
	0	outputs not active	Outputs are set to fallback or freeze values.
	1	outputs active	This bit shall be set, if the corresponding bit 15 in S-0-1500.x.01 IO Control is set and the device has successfully activated its outputs.
14		inputs valid	mandatory
	0	inputs not valid	This bit shall be unset, if the device does not produce valid input data (e.g. caused by a local bus communication error).
	1	inputs valid	This bit shall be set, if the device produces valid input data.
13		Error of resource IO (C1D)	mandatory
	0	no error	
	1	error	cleared with S-0-0099 Reset class 1 diagnostic.
12		Warning of resource IO (C2D)	mandatory
	0	no warning	
	1	warning	This bit disappears automatically, if the reason is no longer present.
11-0		reserved	

Table 12: IO-Status IDN S-0-1500.x.2

3.8.1.2 IO Control

This item corresponds to IDN S-0-1500.x.1 IO Control

Bit No.	Value	Description	Comments
15		output operation state	mandatory
	0	outputs inactive	substitute values activated
	1	outputs activated	
14-0		reserved	

Table 13: IO Control IDN S-0-1500.x

3.8.1.3 RT Data

The following figure will show an example configuration of a FSP IO Slave with 4 Byte Digital Input and 4 Byte Digital Output. The S-Dev/C-Dev and Connection Control are mandatory for each Sercos Slave (FSP IO, FSP Drive or FSP Encoder). The IO Control and IO FGs (Digital Input and Digital Output) are FSP IO specific.

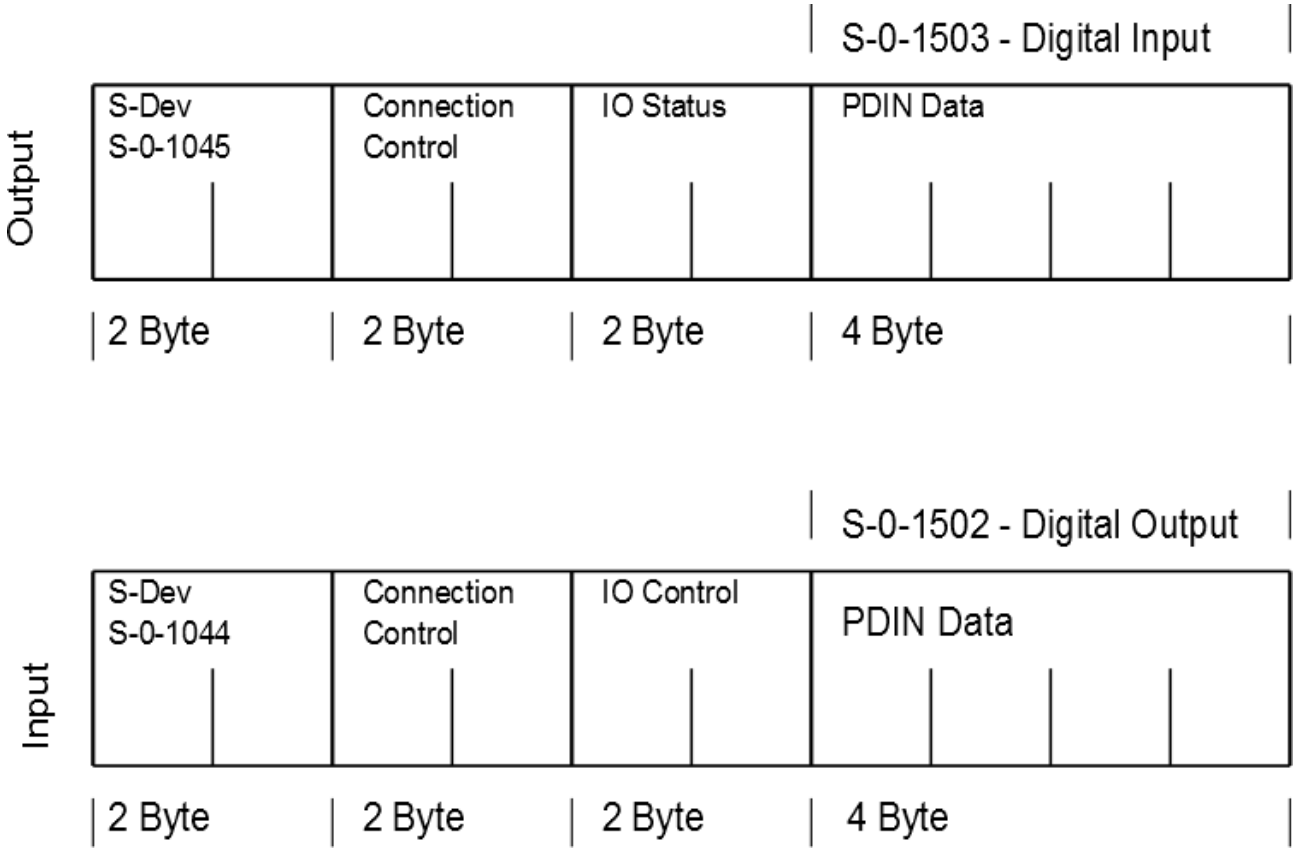


Figure 19: FSP IO RT Data Example

The S-Dev and C-Dev are handled by the stack. The Connection Control can be handled by application or by stack. At LOM projects the application shall handle the Connection Control, IO Status and IO Control. At LFW projects the application can handle the Connection Control, IO Control and IO Status or the stack could do this.

If the stack shall do the handling, set the following flags in the set-configuration packet:

- `SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_HANDLE_CONNCTRL`
- `SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_HANDLE_IOCTRL_IOSTATUS`

3.9 Configuring the TCP Connection using S-0-1048

This IDN is used to activate the three IDNs concerning IP settings (S-0-1020 (IP address), S-0-1021 (Subnet mask) and S-0-1022 (Gateway address)) simultaneously. In order to use it, you first need to provide valid settings to these three standard IDNs.

It is possible to configure default values by set-configuration packet. If the IP address in the set-configuration packet is empty, the default IP address is `192.168.1.<sercos address>`.

Active IP-settings are displayed in IDN S-0-1020.0.1, S-0-1021.0.1 and S-0-1022.0.1.

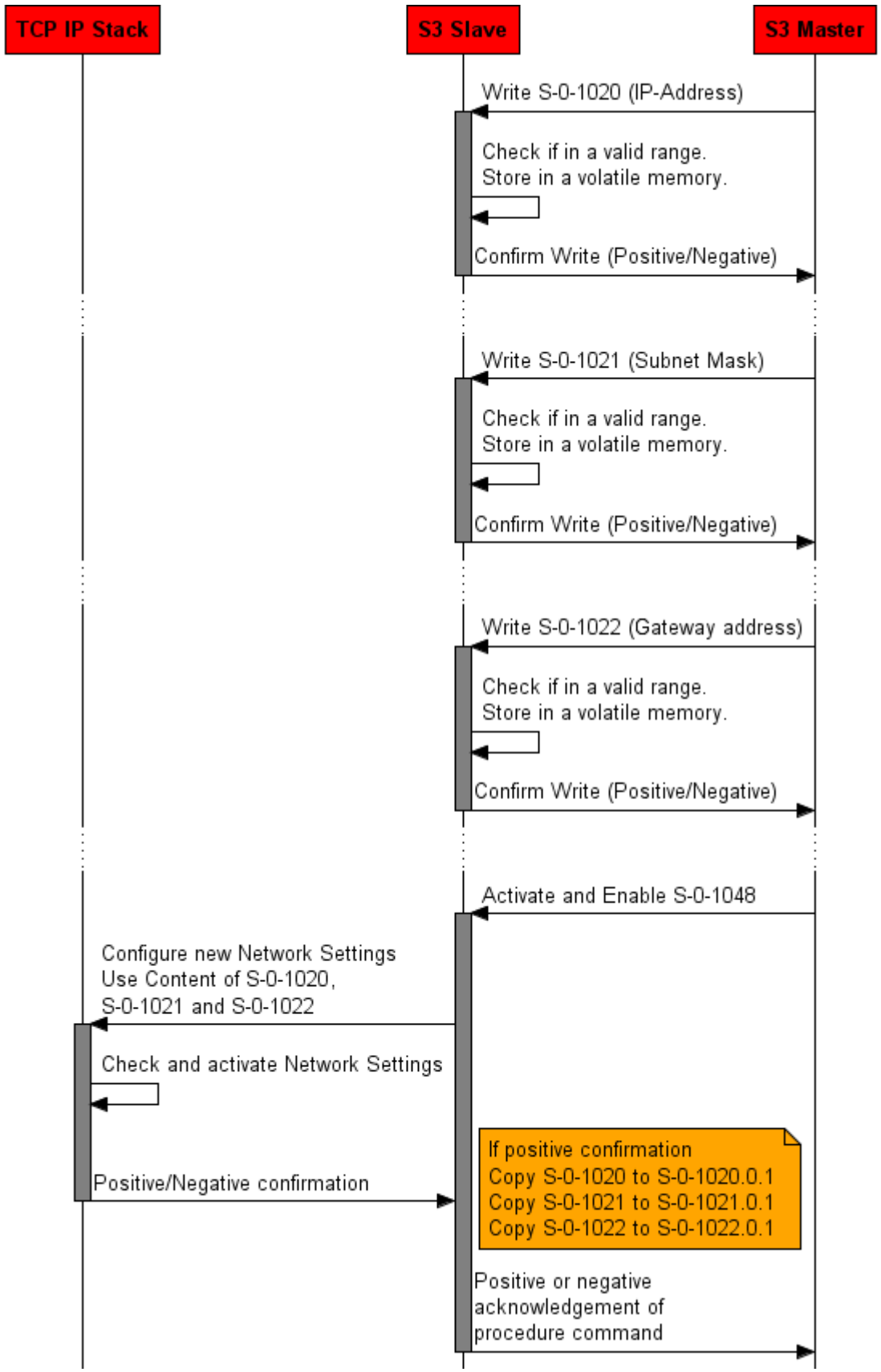


Figure 20: Activate Network Settings

3.10 Diagnosis

The slave provides several built-in diagnosis mechanisms. The state of the device is indicated by the Sercos LED. Within the IDN S-0-0390 (Diagnostic number) the highest diagnostic state can be found. Also a log of the past state changes can be found in IDN S-0-1303.0.10 (Diagnostic Trace buffer).

3.10.1 Sercos LED

Note: The Sercos LED is mandatory. The LED must be labeled with **S**; if not, the device will not receive a Sercos certificate. This LED has three states green, red and orange. Orange must be clearly identifiable. If a duo-LED is used, the certification laboratory will not issue a certificate if just turning on both LEDs. Both colors must be mixed to orange with a light-pipe or a prism.

According to the Sercos recommendations of the General Device Profile, the S LED has the following states:

Pattern	Color	Description	Priority	Comment
#1	dark	NRT-Mode	0	no Sercos communication
#2	orange	CP0	0	communication phase 0 is active
#3		CP1	0	communication phase 1 is active
#4		CP2	0	communication phase 2 is active
#5		CP3	0	communication phase 3 is active
#6	green	CP4	0	communication phase 4 is active
#7		HP0	1	device is in hot-plug phase 0
#8		HP1	1	device is in hot-plug phase 1
#9		HP2	1	device is in hot-plug phase 2
#10		Fast forward & loopback	2	RT-state has changed from fast-forward to loopback
#11		application error	3	see GDP & FSP Status codes class error
#12		MST losses ≥ (S-0-1003/2)	4	as long as the communication warning (S-DEV.Bit15) in the Device Status is present, at least 2 seconds.
#13	red	communication error	5	see SCP Status codes class error
#14		Identification	6	Invoked by (C-DEV.Bit 15 in the Device Control) or SIP Identification request
#15		Watchdog error	7	application is not running
	---------- 3 sec ----------			

Table 14: Sercos LED Definitions

3.10.2 Diagnostic number

The diagnostic number is used to report the highest state in IDN S-0-0390 (Diagnostic Number). Furthermore all Diagnostic numbers are logged within the IDN S-0-1303.0.10 (Diagnostic Trace buffer).

The diagnosis has the following structure:

Bit No.	Value	Description	Comments
31-30		interpretation of bits 29-0	The bits 31-30 define the interpretation of group's source type, class and status codes.
	0	manufacturer specific status codes	bits 29-16 (type and class) defined by Sercos
			bits 15-0 status codes defined by manufacturer
	1	fully manufacturer specific	bits 29-0 are defined by manufacturer
	10	reserved	
	11	Standard	bits 29-0 are defined by Sercos
29-24		source type	
	0x00	FSP Drive	
	0x01	FSP IO	Further detailed IO specific information see S-0-1500.x.32
	0x02	GDP	
	0x03	SCP	
	0x04	CSoS	
	0x05	FSP Encoder	
	0x06	Safety Application	
	0x07...0x3E	reserved	
	0x3F	unknown	
23-20		reserved	
19-16		class	
	0x00-0x09	reserved	
	0x0A	operational state	Priority 4 (lowest).
			Is used to inform about operational state related messages or other information (e.g. Drive HALT, Compatible replacement).
	0x0B	reserved	
	0x0C	procedure command specific state	Priority 3
			Is used to inform about diagnostic events which occur during the execution of a procedure command.
	0x0D	reserved	
	0x0E	warning (C2D)	Priority 2
	0x0F	error (C1D)	Priority 1 (highest)
			Has to be acknowledged using procedure command S-0-0099 Reset class 1 diagnostic.
15-00		Status Code	

Figure 21: Structure of Diagnosis

A list of status codes can be found in the appendix (see section `Other Status` Codes on page 214).

An example for some status codes is the trace buffer (S-0-1303.0.10) after startup:

Status Code	Source type	Class	Message
0xC20AA010	GDP	Operational	The device has been restarted (Power On)
0xC30A0008	SCP	Operational	NRT State
0xC30A0000	SCP	Operational	Communication Phase 0

Figure 22: Example: Status Codes of the Trace Buffer

4 Status Information

The Sercos Slave provides status information in the dual-port memory. The status information has a common block (protocol-independent) and a protocol-specific block (extended status). For the Sercos Slave protocol implementation, the extended status is not used.

For a description of the common status block, see reference [1].

5 The Application Interface

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 is keeping track of all its incoming packets. It provides the communication channel for the underlying Sercos slave Stack. Once, the Sercos slave Stack communication is established, events received by the stack are mapped to packets that are sent to the AP-Task's process queue. On one hand every packet has to be evaluated in the AP-Task's context and corresponding actions be executed. On the other hand, Initiator-Services that are be requested by the AP-Task itself are sent via predefined queue macros to the underlying Sercos slave Stack queues via packets as well.

5.1 Loadable Firmware

5.1.1 Sending Packets

Just sent the packets for the IDN task to `ulDest` = 0x20 (like any other packet). The AP Task will route the packets to the IDN task.

If a Request/Indication was received, the packet must be answered. Copy the content of `tHead` to the Confirmation/Response packet. Modify `ulCmd`, `ulLen` and `ulSta`.

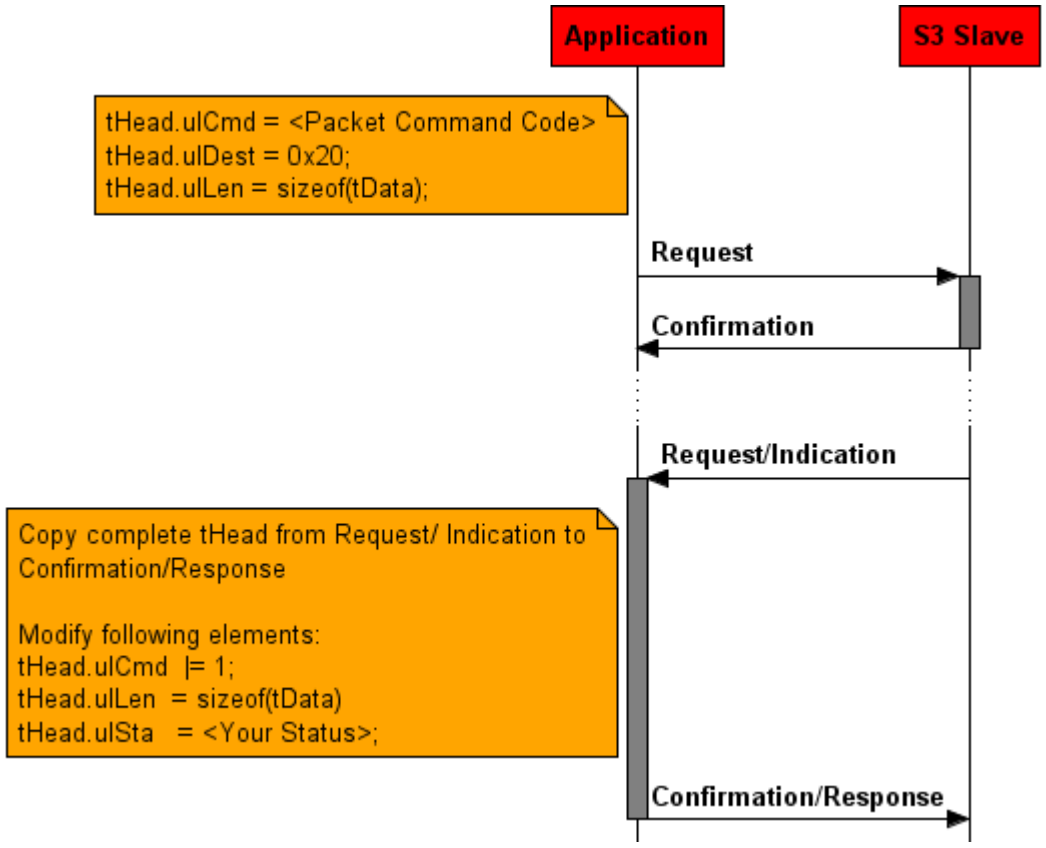


Figure 23: LFW Packet Transfer

5.1.2 The Sercos LED

The Sercos LED will be served by the AP-Task. The application does not need to care.

5.2 Linkable Object Module

5.2.1 Sending Packets

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

ASCII Queue name	Description
`"QUE_S3_SL_IDN"`	Name of the Sercos slave IDN-Task process queue
`"QUE_S3_SL_COM"`	Name of the Sercos Slave COM-Task process queue
`"QUE_S3_SL_NRT"`	Name of the Sercos slave NRT-Task process queue

Table 15: Names of Queues in Sercos Slave Firmware

The returned handles must be used for `ulDest` in all initiator packets to send to the Sercos slave IDN –Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the respective task.

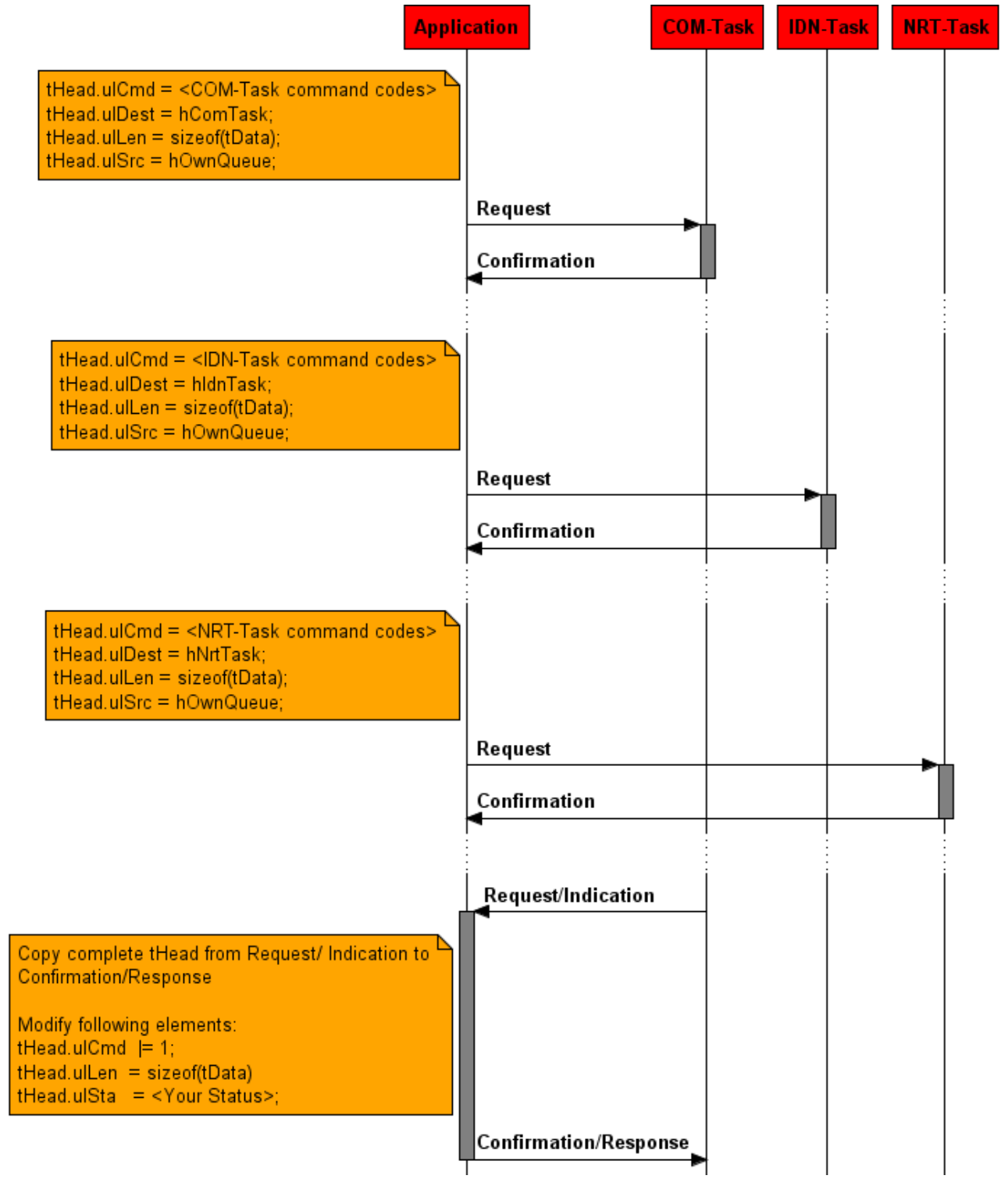


Figure 24: Example of LOM packet transfer

For the destination of each packet use the correct handle. Please use the following table to determine the destination queue of the packet. If the task does not know the packet you will receive the answer “unknown packet”.

Task	Packet
COM-Task	SIII_SL_AP_CMD_SET_CONFIGURATION_REQ
COM-Task	SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_REQ
IDN-Task	SIII_SL_IDN_CMD_CREATE_IDN_REQ
IDN-Task	SIII_SL_IDN_CMD_REGISTER_IDN_NOTIFY_REQ
IDN-Task	SIII_SL_IDN_CMD_WRITE_REQ
IDN-Task	SIII_SL_IDN_CMD_READ_REQ
COM-Task	SIII_SL_COM_CMD_WRITE_DIAGNOSTIC_REQ
COM-Task	SIII_SL_COM_CMD_REMOVE_DIAGNOSTIC_REQ
COM-Task	RCX_REGISTER_APP_REQ
COM-Task	SIII_SL_COM_CMD_PHASE_CHANGE_IND
COM-Task	SIII_SL_COM_CMD_REGISTER_CPX_CHECK_REQ
COM-Task	SIII_SL_COM_CMD_REGISTER_S_0_99_INDICATIONS_REQ
COM-Task	SIII_SL_COM_CMD_UPDATE_DEVICE_STATUS_REQ
NRT-Task	SIII_SL_NRT_SET_CHECK_IP_PARAM_QUEUE_REQ
COM-Task	SIII_SL_COM_CMD_INIT_COMPLETED_IND
COM-Task	SIII_SL_COM_CMD_SET_SERCOS_ADDRESSES_REQ
IDN-Task	SIII_SL_IDN_CMD_MODIFY_MIN_MAX_REQ
IDN-Task	SIII_SL_IDN_CMD_SET_IDN_NAME_REQ
IDN-Task	SIII_SL_IDN_CMD_SET_IDN_UNIT_REQ
IDN-Task	SIII_SL_IDN_CMD_SET_DATA_STATUS_VALID_REQ
IDN-Task	SIII_SL_IDN_CMD_SET_DATA_STATUS_INVALID_REQ
COM-Task	SIII_SL_COM_CMD_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_ INDICATIONS_REQ
COM-Task	SIII_SL_COM_CMD_REGISTER_TRACE_BUFFER_INDICATIONS_REQ

Table 16: Task to packet assignment

5.2.2 The Sercos LED

The Sercos LED shall be handled by your application. To handle the LED a registration for several indications must be done. It is also necessary to implement an own state machine to serve the LED states dependent on their priority.

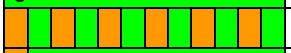
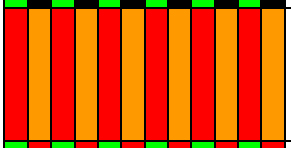
Color	Description	Packet/ Source
dark	NRT-Mode	SIII_SL_COM_CMD_PHASE_CHANGE_IND
orange	CP0	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	CP1	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	CP2	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	CP3	SIII_SL_COM_CMD_PHASE_CHANGE_IND
green	CP4	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	HP0	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	HP1	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	HP2	SIII_SL_COM_CMD_PHASE_CHANGE_IND
	Fast forward & Loopback	SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND
	application error	SIII_SL_COM_CMD_TRACE_BUFFER_UPDATE_IND SIII_SL_COM_CMD_TEST_IDN_DIAGNOSTIC_EVENT_IND SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND Diagnosis occurred in the application.
	MST losses ≥ (S-0-1003/2)	SIII_SL_COM_CMD_TRACE_BUFFER_UPDATE_IND SIII_SL_COM_CMD_TEST_IDN_DIAGNOSTIC_EVENT_IND SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND
red	communication error	SIII_SL_COM_CMD_TRACE_BUFFER_UPDATE_IND SIII_SL_COM_CMD_TEST_IDN_DIAGNOSTIC_EVENT_IND SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND
	Identification	SIII_SL_COM_CMD_CMD_SET_IDENT_LED_IND
	Watchdog error	Must be implemented in Application if needed.
---------- 3 sec ----------		

Figure 25: Codes of the Sercos LED

5.2.3 Data Exchange

Cyclic Data Exchange is controlled by the communication stack. If the stack receives new data or needs new data to send, it calls callback functions. The callbacks must be registered by the application. The following code sample illustrates how to register these callbacks.

```
/* get remote resource handle */
ptRsc->tRem.hInterface = S3s_GetInterface();
if (NULL == ptRsc->tRem.hInterface)
{
  return TLR_DIAG_STA_INIT_REMOTE_FAILED;
}

/* register output buffer callbacks */
if(TLR_S_OK != (tResult = S3Slave_SetupOutputBufferFunctions( ptRsc->tRem.hInterface,
                                                              S3S_RequestOutputBuffer,
                                                              S3S_ReleaseOutputBuffer,
                                                              ptRsc)))
{
  return tResult;
}

/* register input buffer callbacks */
if(TLR_S_OK != (tResult = S3Slave_SetupInputBufferFunctions(  ptRsc->tRem.hInterface,
                                                              S3S_RequestInputBuffer,
                                                              S3S_ReleaseInputBuffer,
                                                              ptRsc)))
{
  return tResult;
}
```

The first parameter is the handle to the remote resources. The second and third parameters are pointers to local callback functions. The last parameter is the pointer to local resources.

5.3 Configuration

5.3.1 Configure the Slave

The SIII_SL_AP_CMD_SET_CONFIGURATION_REQ/CNF service must be used by the AP-Task in order to configure the device with configuration parameters.

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.

A slave stack may use 1 to 8 slave instances. Each slave instance uses one Sercos address and has an own service channel. E.g. this is required for multiple axis per slave device. The master recognizes each axis as a separate slave. The number of slave instances is configured in the variable bNumOfSlaves in the substructure tDeviceCfg. Every slave instance has its own array element in the structure atSlaveCfg[]. If less than 8 slave instances are used, the remaining elements of atSlaveCfg[] remain empty.

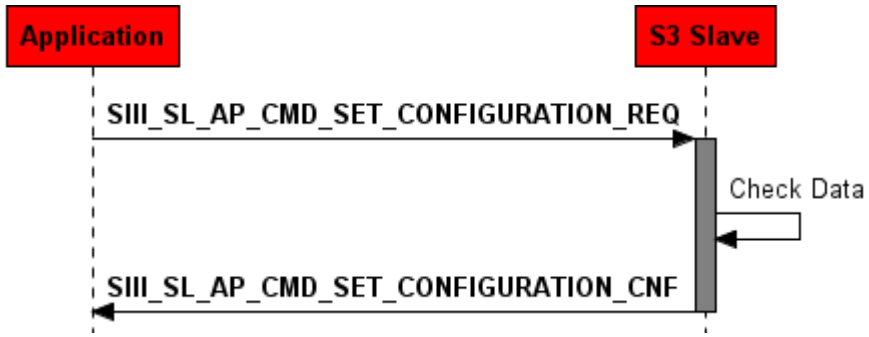


Figure 26: Sequence Diagram for the SIII_SL_AP_CMD_SET_CONFIGURATION_REQ/CNF Packet

Packet Structure Reference

```
typedef struct SIII_SL_AP_SET_CONFIGURATION_REQ_DATA_Ttag
{
  TLR_UINT32                                        ulSystemStart;
  TLR_UINT32                                        ulWdgTime;
  TLR_UINT32                                        ulIOStatus;
  SIII_SL_AP_SET_CONFIGURATION_TCP_DATA_T           tTcpCfg;
  SIII_SL_AP_SET_CONFIGURATION_DEVICE_CFG_DATA_T    tDeviceCfg;
  SIII_SL_AP_SET_CONFIGURATION_SCP_SYNC_DATA_T      tSyncCfg;
  SIII_SL_AP_SET_CONFIGURATION_SLAVE_CFG_DATA_T
atSlaveCfg[SIII_SL_AP_SET_CONFIGURATION_MAX_SLAVES];
} SIII_SL_AP_SET_CONFIGURATION_REQ_DATA_T;

typedef struct SIII_SL_AP_SET_CONFIGURATION_REQ_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  SIII_SL_AP_SET_CONFIGURATION_REQ_DATA_T   tData;
} SIII_SL_AP_SET_CONFIGURATION_REQ_T;
```

Packet Description

Structure SIII_SL_AP_SET_CONFIGURATION_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_AP	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY( ): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	968	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x3628	SIII_SL_AP_CMD_SET_CONFIGURATION_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_AP_SET_CONFIGURATION_REQ_DATA_T			
ulSystemStart	UINT32	0,1	Behavior at system start (bit field): 0 = automatic (default) 1 = application controlled
ulWdgTime	UINT32	0, 20..65535	Watchdog time specified in ms: 0 = Watchdog timer has been switched off 1000 = default value
ulIOStatus	UINT32		Input/Output Status (currently not used)
tTcpCfg	structure SIII_SL_AP_SET_CONFIGURATION_TCP_DATA_T TCP/IP configuration data, see Table 18: TCP/IP Configuration data - parameters, their meanings and their ranges of allowed values		
tDeviceCfg	structure SIII_SL_AP_SET_CONFIGURATION_DEVICE_CFG_DATA_T Device configuration, see Table 19: Device Configuration data - parameters, their meanings and their ranges of allowed values		
tSyncCfg	structure SIII_SL_AP_SET_CONFIGURATION_SCP_SYNC_DATA_T SCP_Sync configuration, see Table 21: SCP_Sync Configuration data - parameters, their meanings and their ranges of allowed values		
atSlaveCfg[8]	structure SIII_SL_AP_SET_CONFIGURATION_SLAVE_CFG_DATA_T Slave configuration, see Table 22: Slave Configuration data - parameters, their meaning and their ranges of allowed values		

Table 17: SIII_SL_AP_CMD_SET_CONFIGURATION_REQ – Set Configuration Request

The TCP/IP configuration data structure SIII_SL_AP_SET_CONFIGURATION_TCP_DATA_T is related to parameter tTcpCfg. It consists of the following items:

Parameter	Type	Meaning	Range of Values
ulTcpFlag	UINT32	TCP flag	0 … 0x1F
ulIpAddr	UINT32	IP address	Default value: 0.0.0.0
ulNetMask	UINT32	Net mask	Default value: 0.0.0.0
ulGateway	UINT32	Gateway	Default value: 0.0.0.0

Table 18: TCP/IP Configuration data - parameters, their meanings and their ranges of allowed values

The bits of the TCP flag indicate whether the following items are present (bit set) or not (bit not set):

- Bit 0: IP Address
- Bit 1: Net Mask
- Bit 2: Gateway
- Bit 4: DHCP

To use a fix IP address, set bit 0 to 1 and deactivate DHCP (bit 4 = 0). To use a IP address from a DHCP server, activate DHCP (bit 4 = 1) and set bit 0, 1, and 2 to 0. If bit 1 is set to 1 and bit 4 is set to 1, then the stack uses DHCP first to obtain an IP address from a DHCP server.

The device configuration structure SIII_SL_AP_SET_CONFIGURATION_DEVICE_CFG_DATA_T is related to parameter tDeviceCfg. It consists of the following items:

Parameter	Type	Meaning	Range of Values
bProcDataMode	UINT8	see Table 20 on page 64	0..2
bProcDataOutputUpdateMode	UINT8	Not used yet. Process Data Output Update Mode.	0
bProcDataInputUpdateMode	UINT8	Not used yet. Process Data Input Update Mode.	0
bNumOfSlaves	UINT8	Number of slaves	1..8
bSCP_Sync_Version	UINT8	Version number of profile SCP_Sync	0..3
bSCP_SIP_Version	UINT8	Version number of profile SCP_IP	0..1
bSCP_NRT_Version	UINT8	Version number of profile SCP_NRT see Chapter 2.4 0: No NRTPC 1: NRTPC v1 2: NRTPC v2. The stack creates IDNs listet in Table 117 (page 227). **Note:** Requirement to the application: The application must store the IP address non-volatile and use it to configure the slave.	0..2
bSercosVersion	UINT8	Stack runs compatible to the configured Sercos Version. This setting is necessary to pass the Sercos Conformizer test according Sercos specification V1.1.2 or V1.3.1. 0: V1.1.2 1: V1.3.1	0..1

ausReserved[3]	UINT16	Reserved field	0
usVendorCode	UINT16	Vendor code: as defined in IDN S-0-1300.x.03. Use 1000 (0x3e8) for “Hilscher GmbH”. Other vendor codes are assigned by SERCOS International.	1..65535
abDeviceId[256]	UINT8	Device ID as defined in IDN S-0-1300.x.05.	This is a NUL-terminated string
ulInputDataErrorThreshold	UINT32	Not used yet.	0
ulOutputDataErrorThreshold	UINT32	Not used yet.	0
ulSyncErrorThreshold	UINT32	Not used yet.	0

Table 19: Device Configuration data - parameters, their meanings and their ranges of allowed values

Parameter bProcDataMode

Value	Description
0x0	SIII_SL_AP_SET_CONFIGURATION_DEVICE_PROCDATA_MODE_BUS_SYNC Application works synchronous to the bus cycle. See section 3.3.1.1”Bus Synchronous ” on page 29 for more details.
0x1	SIII_SL_AP_SET_CONFIGURATION_DEVICE_PROCDATA_MODE_TRIPLE_BUFFER Use Triple buffer at AP task. Use this configuration if the application is running acyclic to bus cycle.
0x2	SIII_SL_AP_SET_CONFIGURATION_DEVICE_PROCDATA_MODE_SEPARATE_AT_CALLBACK Application works synchronous to the bus cycle. Stack has a separated AT callback to update IO Data short before ATs arrive. This mode allows a faster data transmission from slave to the master. See section 3.3.1.2”Bus Synchronous Mode with reduced round-trip time” on page 30 for more details.

Table 20: bProcDataMode configuration variants

The SCP_Sync configuration structure `SIII_SL_AP_SET_CONFIGURATION_SCP_SYNC_DATA_T` is related to parameter `tSyncCfg`. It consists of the following items:

Parameter	Type	Meaning	Range of values
ulSyncFlags	UINT32	Sync Configuration Flags Bit 0: Con_Clk output enable: 0 – disabled; 1 – enabled Bit 1: Con_Clk output active level: 0 - low active; 1 - high active Bit 2: Div_Clk output enable: 0 – disabled; 1 – enabled Bit 3: Div_Clk output active level: 0 - low active; 1 - high active Bit 4: Div_Clk mode: 0 - Mode 0; 1 - Mode 1 (see below) Bit 5-15 : reserved	
ulConClkLength	UINT32	Controller Clock Length in ns	1000..655350
ulTDivClk	UINT32	Time Divided Control Clock in ns	0..0xFFFFFF
ulDTDivClk	UINT32	since V3.1.X: unused	0..0x3FFFFFF
ulDivClkLength	UINT32	since V3.1.X: unused, always fixed 1 µs	1000..20000
ulNDivClk	UINT32	number of pulses (mode 0) or cycles (mode 1)	1..255
ulTInt2	UINT32	unused	0
ulTInt3	UINT32	unused	0

Table 21: SCP_Sync Configuration data - parameters, their meanings and their ranges of allowed values

All timing values are handled as multiples of 10 ns (e. g. `ulConClkLength` = 1005 ns is handled as 1000 ns). The maximum valid value of `ulConClkLength` depends on the configured cycle time. In general: the signal must go to inactive again before the next cycle starts.

Note: Since version V3.1.X the Sync configuration has changed. The parameters `ulDivClkLength` and `ulDTDivClk` are not evaluated any more. In Div_Clk mode 0 it is not possible any more to configure aperiodic signals. `ulConClkLength` has now a maximum of 655350 ns.

Div_Clk mode 0 – N-times within one communication cycle

Div_Clk signal becomes active several times within a communication cycle. The delay time for the first pulse is provided with `ulTDivClk` parameter. The number of pulses within a communication cycle is provided with `ulNDivClk`. The pulse length of the Div_Clk is provided with `ulDivClkLength`. The distance between two Div_Clk pulses is calculated: bus cycle time (CP4) divided by `ulNDivClk`. Since V3.1.X it is absolutely necessary, that there is no division remainder! E. g. 1ms bus cycle time with N=2 or 4 will work. But N=3 does not work (333.333 ns * 3 != 1.000.000 ns). As consequence no Con_Clk and Div_Clk signals are generated. The following figure shows a configuration with `ulNDivClk` = 5.

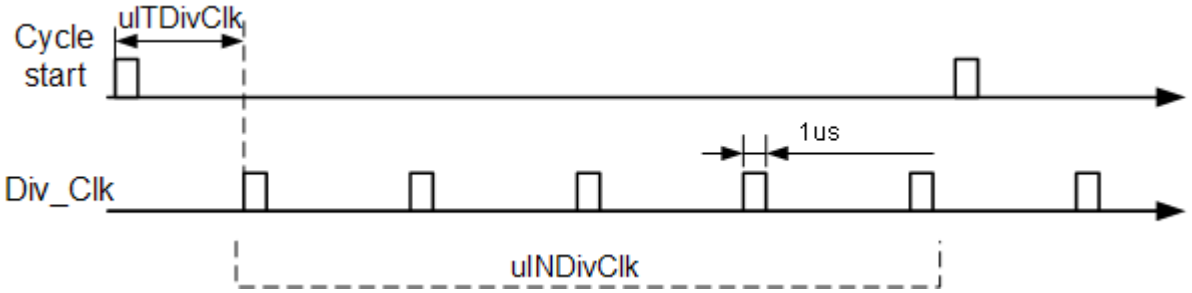


Figure 27: Div_Clk in mode 0

Div_Clk mode 1 – once after N communication cycles

Div_Clk signal becomes active once after N communication cycles. The delay time for the first pulse is provided with `ulTDivClk` parameter. The number of communication cycles is provided with `ulNDivClk`. Since V3.1.X the pulse length of the Div_Clk is always 1 µs.

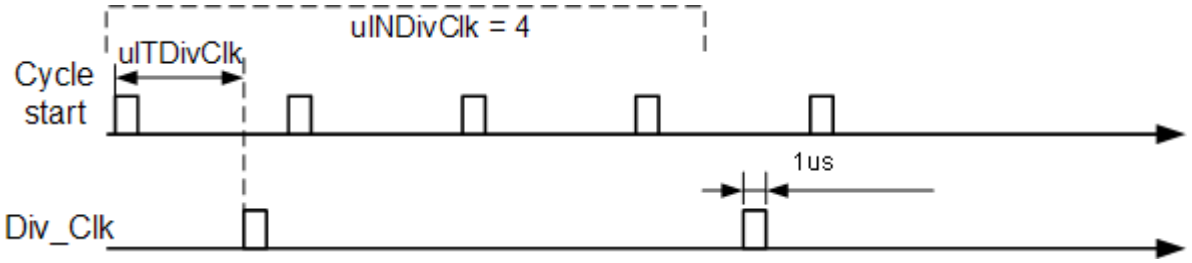


Figure 28: Div_Clk in mode 1

The slave configuration structure `SIII_SL_AP_SET_CONFIGURATION_SLAVE_CFG_DATA_T` related to parameter field `atSlaveCfg[]` describes the slaves and their connections. The number of connections per slave is limited to 4. The slave configuration structure consists of the following items:

Parameter	Type	Meaning	Range of Values
usSERCOSAddress	UINT16	Sercos Address	0..511
usSlaveFlags	UINT16	See *Table 23: Explanation of usSlaveFlags within Slave Configuration Data*	see below
usSCP_CfgType	UINT16	SCP config type according to Sercos specification: 0x0101 - SCP_FixCFG (latest Stack Version of FixCFG) 0x0201 - SCP_VarCFG Version 1.1.1 0x0202 - SCP_VarCFG Version 1.1.3	0x0101, 0x0201, 0x0202
ulFSP_Type	UINT32	FSP type 0x10001 - FSP_IO 0x20001 - FSP_DRIVE 0x30001 – FSP Encoder	 0x10001 0x20001 0x30001
usSCP_FixCfg_OutputDataSize	UINT16	Output Data Size (the slave consumes this data) for fixed configuration (SCP_FixCfg) only. **Note**: This value must be even. The maximum value for usSCP_FixCfg_OutputDataSize depends on the number of used slaves. Each slave uses 2 bytes for connection control and 2 bytes for IO control. netX 100/netX 500: max = 124 - 4 * bNumOfSlaves netX 50/netX 51/netX 52: max = 276 - 4 * bNumOfSlaves	0, 2, 4, ... max
usSCP_FixCfg_InputDataSize	UINT16	Input Data Size (the slave produces this data) for fixed configuration (SCP_FixCfg) only. **Note**: This value must be even. The maximum value for usSCP_FixCfg_InputDataSize depends on the number of used slaves. Each slave uses 2 bytes for connection control and 2 bytes for IO status. netX 100/netX 500: max = 132 - 4*bNumOfSlaves netX 50/netX 51/netX 52: max = 284 - 4*bNumOfSlaves	0, 2, 4, ... max
ausUserSCP_Types[20]	UINT16[]	Up to 20 User SCP Types. A value of 0x0 marks the end of the list. Following User SCP types are already defined: 0x0401 - SCP_WD Version 1.1.1 for monitoring connections 0x0501 - SCP_Diag Version 1.1.1 for bus-diagnosis 0x0601 - SCP_RTB Version 1.1.1 for using Real time bits 0x0901 - SCP_Mux Version 1.1.1 for multiplexed cyclic data 0x0B01 - SCP_Sig Version 1.1.1 for using signal status/control words	
atConnDataOffsets[4]	See below	Connection Data Offsets (for 4 connections at maximum) **Note**: Each operation data shall start at an **even** number of octets within the connection.	see below

Table 22: Slave Configuration data - parameters, their meaning and their ranges of allowed values

Parameter usSlaveFlags

Bit	Description
0	SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_ADDRESS_IS_NOT_CHANGEABLE 0 -SERCOS address may be changed by the master 1 - SERCOS address may not be changed by the master
1	SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_DELETE_NON_STACK_OBJECTS 0 - Non stack objects are not deleted on Channel Init 1 - Non stack objects are deleted on Channel Init
2	SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_SETUP_DEFAULT_OD 0 - Do not set up a default OD 1 - Set up a default OD only available for SCP_FixCFG (recommended for SCP_FixCFG) (this implies SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_DELETE_NON_STACK_OBJECTS)
3	SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_HANDLE_CONNCTRL 0 – Handling of connection control is application controlled 1 – Handling of connection control is stack controlled
4	SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_HANDLE_IOCTRL_IOSTATUS 0 – Handling of IO control / IO status is application controlled 1 – Handling of IO control / IO status is stack controlled This flag can only be set for SCP_FixCFG in combination with FSP IO
5-15	Reserved

Table 23: Explanation of `usSlaveFlags` within Slave Configuration Data

Parameter atConnDataOffsets

For more information on SCP Profiles (SCP config, User SCP) see section 4 (*SERCOS Communication Profiles Classes)* of the SERCOS Communication Profile V1.3 specification.

The parameter field `atConnDataOffsets[4]` provides a structure for the administration of connections of the Sercos Slave. The entire contents of the parameter field relates to the dual-port memory. In detail,

- `atConnDataOffsets[0]` is related to the slave-to-master communication (in the producer role).
- `atConnDataOffsets[1]` is related to the master-to-slave communication (in the consumer role).
- `atConnDataOffsets[2]` is related to the slave-to-slave communication (in the producer role).
- `atConnDataOffsets[3]` is related to the slave-to-slave communication (in the consumer role).

Each such structure of the array `atConnDataOffsets[4]` looks like:

Parameter	Type	Meaning	Range of Values
usConnCtrlOffset	UINT16	Connection Control offset specifying the position of the offset of the connection control process image within the DPM.	0…5758
usRtDataOffset	UINT16	Real time data process image offset	See notes 1, 2
usRtDataMaxLength	UINT16	Real time data maximum length. This value must be even.	See notes 1, 2 and 3

Table 24: `SIII_SL_AP_SET_CONFIGURATION_CONN_CFG_DATA_T` – parameters, their meanings and their ranges of allowed values

For the range of values of the parameters `usRtDataOffset` and `usRtDataMaxLength` the following applies:

- Note 1: The lower limit for each of `usRtDataOffset` and `usRtDataMaxLength` is 0.
- Note 2: The sum of `usRtDataOffset` and `usRtDataMaxLength` must be less than or equal to 5760.
- Note 3: Compute the value of `usRtDataMaxLength` according to the following rules:

  The overall sum of the input data of all connections of all slaves must not exceed the given length in the technical data.

Note: The given byte length includes pure IO-Data, connection control and IO-status/control. In total, you get a 4 byte overhead for each slave.

If your configured process data image does not fit, you will receive the communication error `TLR_E_SERCOSIII_SL_COM_INVALID_PROCESS_DATA_IMG(0xC04E003FL)`

Simple Configuration Example for 2 Slaves

The following example explains how to configure two slaves both for the SCP profile FixCfg 1.1.1 and the FSP profile FSP IO.

- Slave 1 (i.e. Sercos address 1) is set to an input data size of 4 and an output data size of 6.
- Slave 2 (i.e. Sercos address 2) is set to an input data size of 6 and an output data size of 8.

In order to achieve this, configure these slaves as follows:

Slave 1

```
atSlaveCfg[0].usSercosAddress = 1         /* Sercos address 1 */
atSlaveCfg[0].usSCP_CfgType = 0x0101      /* SCP profile FixCfg 1.1.1 */
atSlaveCfg[0].ulFSP_Type = 0x1001         /* FSP profile FSP IO */
atSlaveCfg[0].usFixCfg_OutputDataSize = 6 /* OutputDataSize for Slave 1 */
atSlaveCfg[0].usFixCfg_InputDataSize = 4  /* InputDataSize for Slave 1 */
```

Slave 2

```
atSlaveCfg[1].usSercosAddress = 2         /* Sercos address 2 */
atSlaveCfg[1].usSCP_CfgType = 0x0101      /* SCP profile FixCfg 1.1.1 */
atSlaveCfg[1].ulFSP_Type = 0x1001         /* FSP profile FSP IO */
atSlaveCfg[1].usFixCfg_OutputDataSize = 8 /* OutputDataSize for Slave 2 */
atSlaveCfg[1].usFixCfg_InputDataSize = 6  /* InputDataSize for Slave 2 */
```

Next, the connections need to be configured. Have in mind, that every connection needs a 2 byte Connection Control (see section *Connection Control* on page 40) and 2 byte IO Control.

Slave 1

For the slave-to-master communication, `atConnData.Offsets[0]` has to be set as follows:

```
atSlaveCfg[0].atConnData.Offsets[0].usConnCtrlOffset = 0  /* Offset Conn-Ctrl Input */
atSlaveCfg[0].atConnData.Offsets[0].usRtDataOffset = 2
atSlaveCfg[0].atConnData.Offsets[0].usRtDataMaxLength = 6
```

For the master-to-slave communication, `atConnData.Offsets[1]` has to be set as follows:

```
atSlaveCfg[0].atConnData.Offsets[1].usConnCtrlOffset = 0  /* Offset Conn-Ctrl Output */
atSlaveCfg[0].atConnData.Offsets[1].usRtDataOffset = 2
atSlaveCfg[0].atConnData.Offsets[1].usRtDataMaxLength = 8
```

Slave 2

For the slave-to-master communication, `atConnData.Offsets[0]` has to be set as follows:

```
atSlaveCfg[1].atConnData.Offsets[0].usConnCtrlOffset = 8  /* Offset Conn-Ctrl Input */
atSlaveCfg[1].atConnData.Offsets[0].usRtDataOffset = 10
atSlaveCfg[1].atConnData.Offsets[0].usRtDataMaxLength = 8
```

For the master-to-slave communication, `atConnData.Offsets[1]` has to be set as follows:

```
atSlaveCfg[1].atConnData.Offsets[1].usConnCtrlOffset = 10 /* Offset Conn-Ctrl Output */
atSlaveCfg[1].atConnData.Offsets[1].usRtDataOffset = 12
atSlaveCfg[1].atConnData.Offsets[1].usRtDataMaxLength = 10
```

The offset values for connection control for input and output are simply calculated by adding the respective values for the size of the Connection Control (`usRtDataOffset`) of slave 1 and the maximum size of the real time data area of slave 1 (`usRtDataMaxLength`) of slave 1.

The offset value for the real time data is 2 (size of the Connection Control) for the first slave. For the second *slave*, this value is calculated as the sum of 2 (size of the Connection Control) and the offset value for connection control for input or output, respectively.

The maximum size of the real time data area for input or output is calculated as the sum of 2 (size of IO Control) and the respective input or output data size of the second slave.

All other members of `atSlaveCfg[0]` and `atSlaveCfg[1]` can be set according to your needs for the specific slave.

The upper part of *Figure 29* below illustrates the entire parameter assignment in detail. The lower part explains the order and size of Connection Control, IO Control and real time data area for input and output.

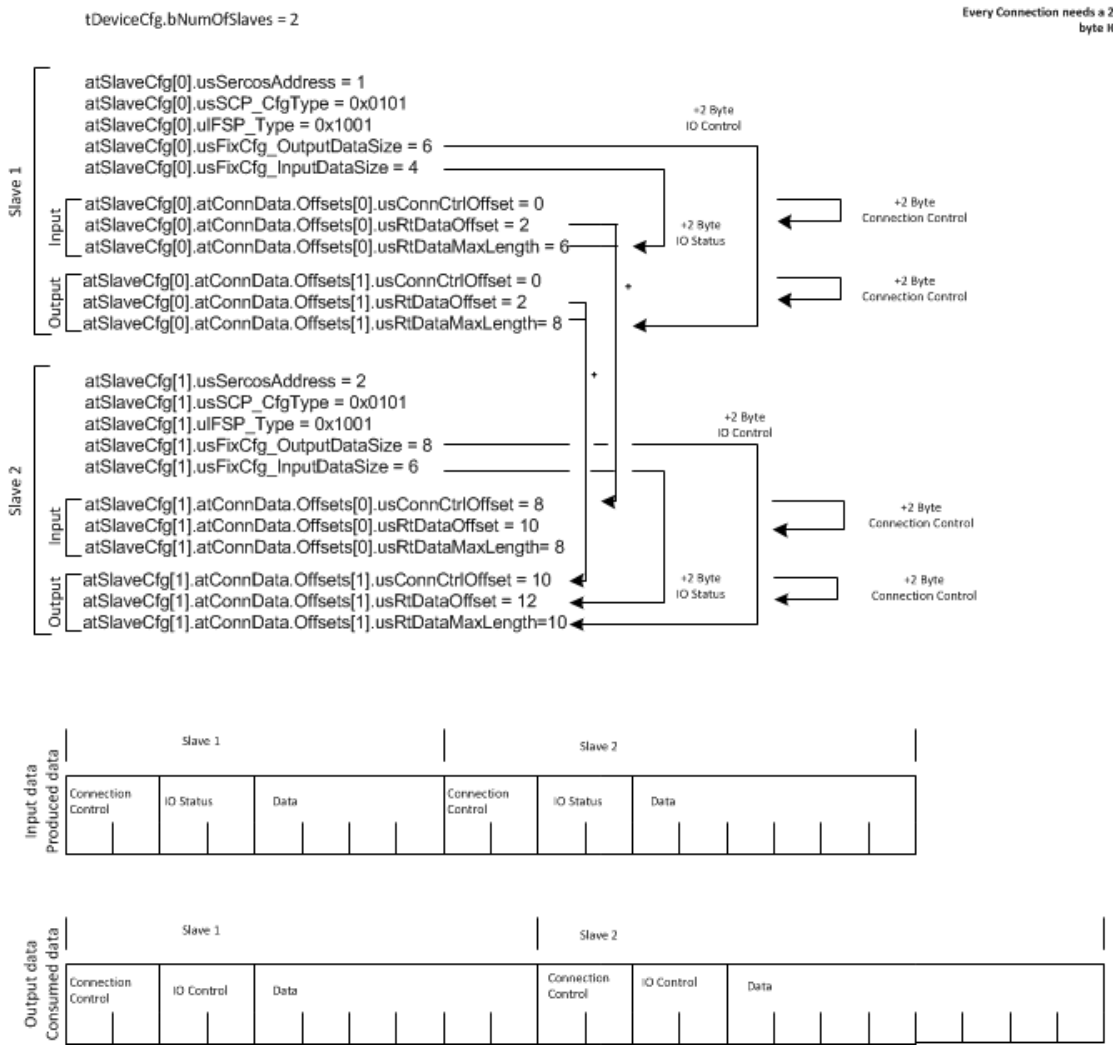


Figure 29: Simple Configuration Example for 2 Slaves

Packet Structure Reference

```
typedef struct SIII_SL_AP_SET_CONFIGURATION_CNF_Ttag
{
  TLR_PACKET_HEADER_T                   tHead;
} SIII_SL_AP_SET_CONFIGURATION_CNF_T;
```

Packet Description

Structure SIII_SL_AP_SET_CONFIGURATION_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3629	SIII_SL_AP_CMD_SET_CONFIGURATION_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change

Table 25: SIII_SL_AP_CMD_SET_CONFIGURATION_CNF – Confirmation of Set Configuration request

5.3.2 Reconfigure Connections

You can use the packet SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_REQ in order to remove, add or change connections and their offset data.

Note: This packet may just be used after the packet SIII_SL_AP_SET_CONFIGURATION_REQ and before RCX_CHANNEL_INIT_REQ, see *Figure 30: Change Connections* below. The slave must be configured to support SCP VarCfg (usSCP_CfgType).

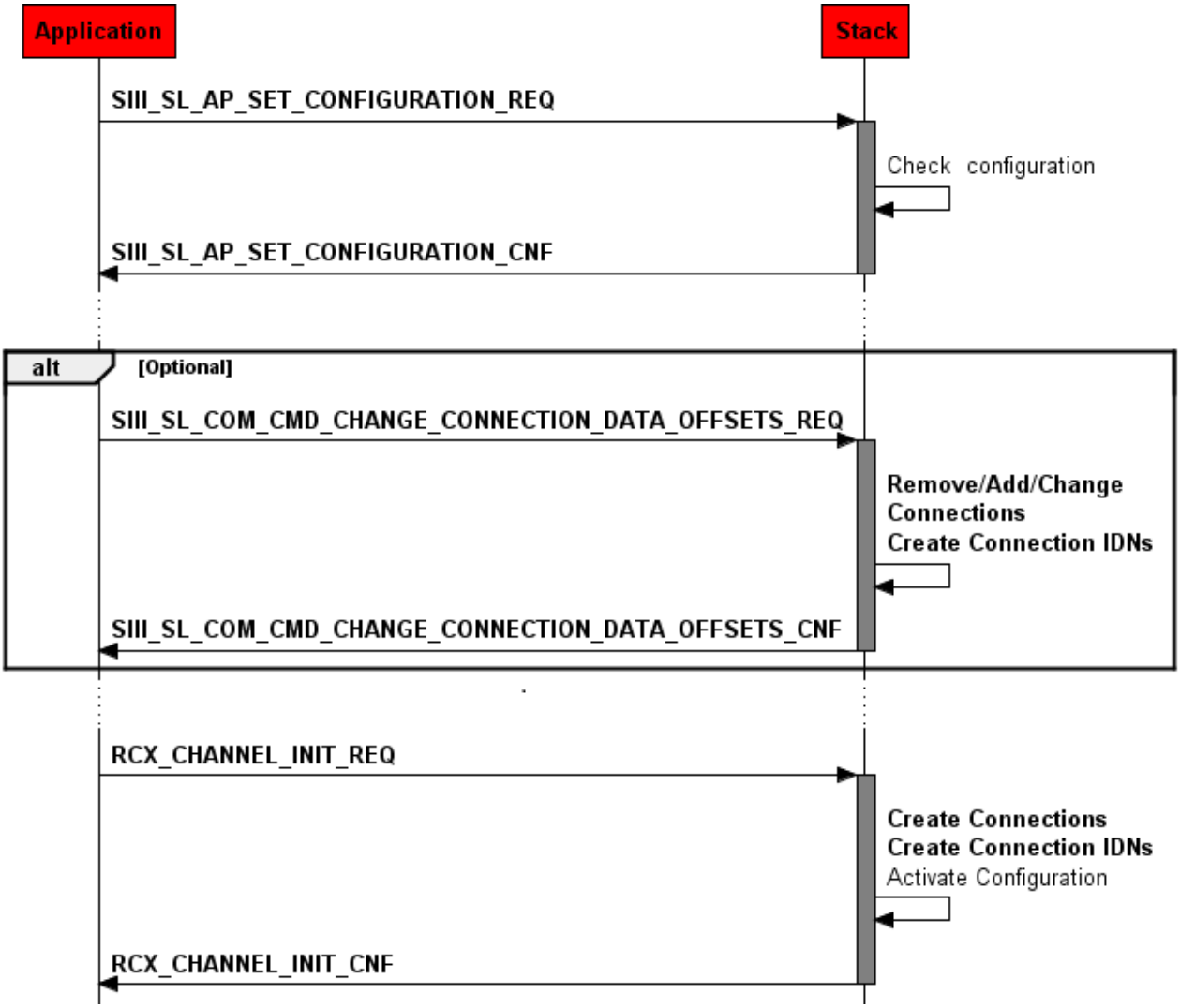


Figure 30: Change Connections

If the application has removed, added or changed the connection configuration, the stack does the following after a channel init:

- If additional connections are created the stack creates the necessary IDNs.
- If the application has requested to remove all previously created connections (bFlag = SIII_SL_COM_CHANGE_CONNECTION_DATA_REMOVE_CONNECTIONS_PREVIOUSLY), then the stack removes the IDNs of these connections. In consequence, a **registration** for the IDNs will also **be removed** and the application **must register** for this IDNs **again** using SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ.

In most cases it is not necessary to remove all previously created connections (`bFlags` set to 0x1). This is just necessary, if less than 4 connections per slave shall be created, because as default the Set-Configuration packet creates 4 connections per slave.

If the previous connections are removed, also the respective connection IDNs (S-0-1050.x.x) will be deleted. This will also **remove registrations** for these IDNs.

During IDN S-0-0127 it is possible to reconfigure the connection by this packet. But if doing so, it is not allowed to use the reconfiguration mechanism in the S-0-0127 indication packet. The packet `SIII_SL_COM_CMD_PROC_CMD_S_0_0127_EXEC_IND` must be answered within 5 seconds. Do not change the amount of connections during S-0-0127, just change the offsets etc.

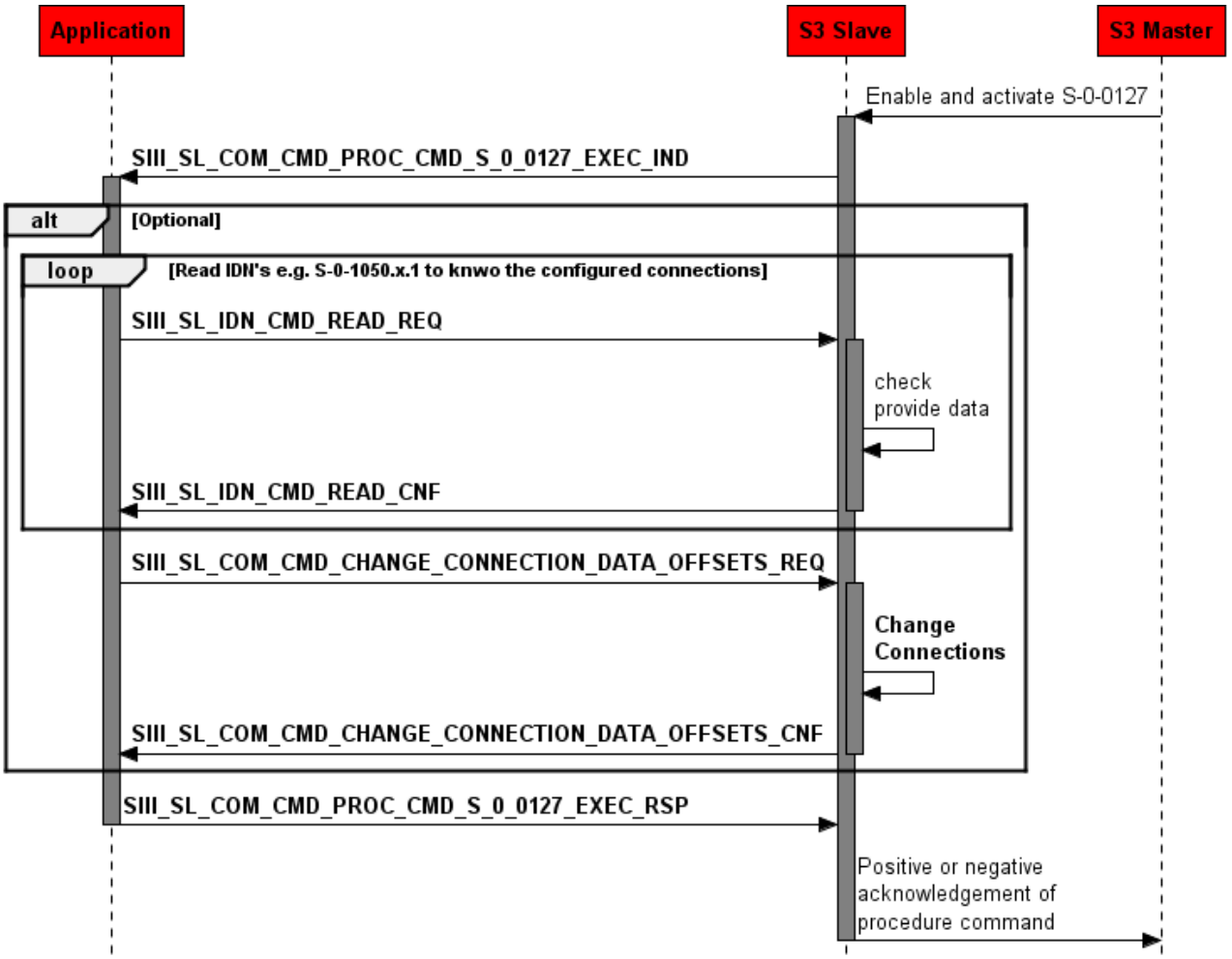


Figure 31: Connection reconfiguration during S-0-0127

For a single slave device it is possible to configure up to 60 connections. The number of connections depends on the configured slaves. Each configured slave decreases the number of available connections.

Slaves in Device	Sum of available Connections
1	60
2	56
3	52
4	48
5	44
6	40
7	36
8	32

Table 26: Number of available connections depending on number of configured slaves

The minimum connection length for an IO Device is 6 Byte i.e.
2 Byte Connection Control + 2 Byte IO Status/IO Control + 2 Byte minimum RT Data length.

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct
SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CONFIG_DATA_REQ_Ttag
{
  uint8_t                                              bSlaveIdx;
  uint8_t                                              bConnectionIdx;
  uint16_t                                             usConnCtrlOffset;
  uint16_t                                             usRtDataOffset;
  uint16_t                                             usRtDataMaxLength;

}SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CONFIG_DATA_REQ_T;

typedef __TLR_PACKED_PRE struct SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_REQ_Ttag
{
  uint8_t bFlags;
  uint8_t bNumberOfConnections;
  SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CONFIG_DATA_REQ_T atConfiguration[64];
} __TLR_PACKED_POST SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_REQ_T;

typedef struct SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                  tHead;
  SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_REQ_T  tData;
} SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_REQ_T;

#define SIII_SL_COM_CHANGE_CONNECTION_DATA_REMOVE_CONNECTIONS_PREVIOUSLY
0x00000001
```

Packet Description

Structure SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	20+n	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32B8	SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_REQ_T			
bFlags	UINT8	0..1	Index of the slave device 0x1 : delete all previously configured connections and create all connection new 0x0 : update previously configured connections (just change values)
bConnectionIdx	UINT8	0..60	Number of configured connections in the following array
atConfiguration[60]	See below		See: SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CONFIG_DATA_REQ_T

Table 27: SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_REQ – Change Connections Request

tData - Structure SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CONFIG_DATA_REQ_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
bNumberOfConnections	UINT8	0..60	Connection Index
usConnCtrlOffset	UINT16	0..240	Connection Control offset specifying the position of the offset of the connection control process image within the DPM. See
usRtDataOffset	UINT16	0..48	Real time data process image offset
usRTDataMaxLength	UINT16		Real time data maximum length. This value must be even.

Table 28: SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CONFIG_DATA_REQ_T – Configure new connections

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_CNF_Ttag
{
  TLR_UINT8                                                bSlaveIdx;
} __TLR_PACKED_POST SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_CNF_T;

typedef struct SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                      tHead;
  SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_CNF_T  tData;
} SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CNF_T;
```

Packet Description

Structure SIII_SL_COM_CHANGE_CONNECTION_DATA_OFFSETS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32B9	`SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_CNF` - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_ COM_CHANGE_CONNECTION_DATA_OFFSETS_DATA_CNF_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 29: `SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_CNF` – Change Connection Confirmation

5.4 IDN Handling - Required Packets

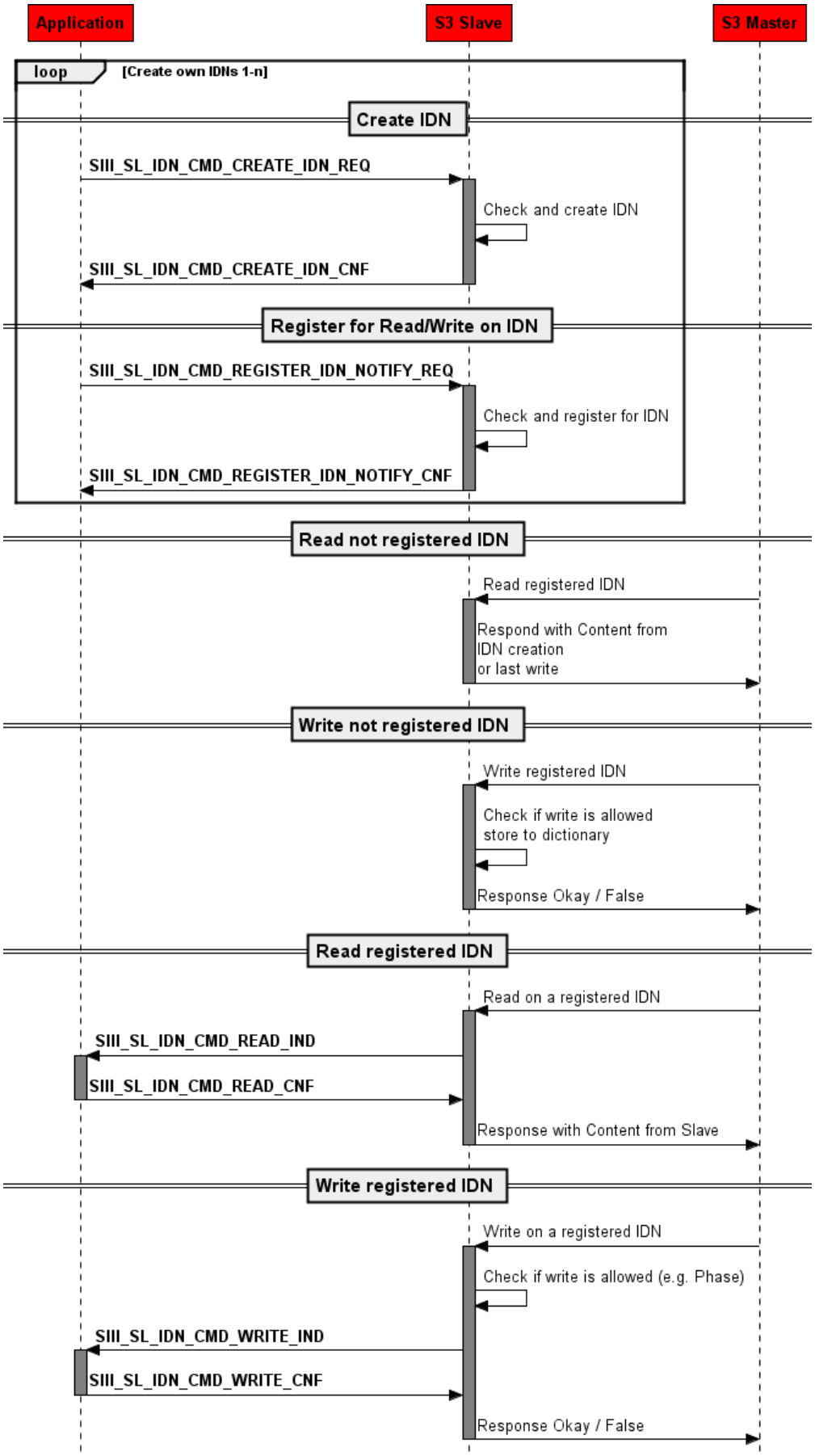


Figure 32: Basic IDN packets

5.4.1 Create IDN Request

The packet `SIII_SL_IDN_CMD_CREATE_IDN_REQ/CNF` creates an IDN within the object dictionary.

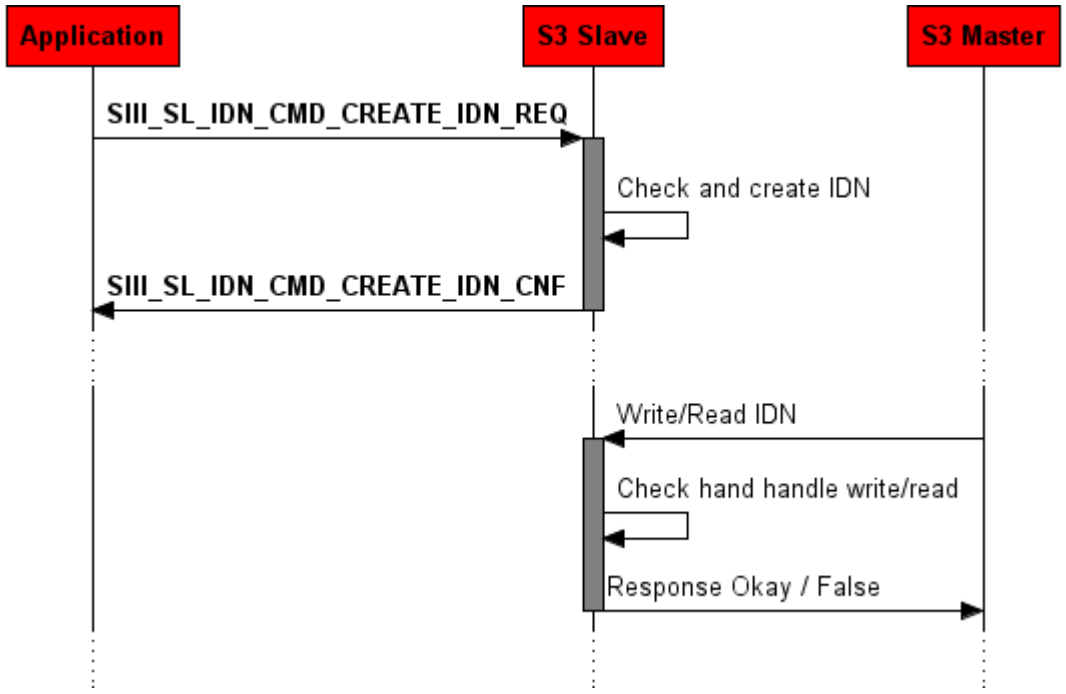


Figure 33: Sequence Diagram for the SIII_SL_IDN_CMD_CREATE _REQ/CNF Packet

The IDN to be accessed is specified in variable `ulIdn`.

Note: You must not specify IDNs which have already been created by the stack. If you nevertheless try doing this, an error will be reported. For a list of IDNs, which are created by the stack, see section 8.2.

The slave to be accessed is specified in variable `bSlaveIdx`.

The bit field `bValueInfo` defines which initial values are appended in `abData`. The order of the optional initial values is:

1. DATASTATUS
2. NAME
3. UNIT
4. MINIMUM
5. MAXIMUM
6. VALUE

Packet Structure Reference

```
typedef struct SIII_SL_IDN_CREATE_IDN_REQ_DATA_Ttag
{
  /* unfragmentable part */
  TLR_UINT32                          ulIdn;
  TLR_UINT8                           bSlaveIdx;
  TLR_UINT8                           bValueInfo;
  TLR_UINT16                          usMaxNameLength;
  TLR_UINT16                          usMaxUnitLength;
  TLR_UINT16                          usMaxListDataSize;
  TLR_UINT32                          ulAttribute;
  TLR_UINT32                          ulTotalLength;
  /* fragmentable part */
  TLR_UINT8                           abData[1];
} SIII_SL_IDN_CREATE_IDN_REQ_DATA_T;

#define SIII_SL_IDN_CREATE_IDN_MIN_DATA_SIZE (sizeof(SIII_SL_IDN_CREATE_IDN_REQ_DATA_T) -
sizeof(((SIII_SL_IDN_CREATE_IDN_REQ_DATA_T*)0)->abData))

#define MSK_SIII_SL_IDN_CREATE_IDN_VALUE_INFO_DATASTATUS 0x01  /* initial value */
#define MSK_SIII_SL_IDN_CREATE_IDN_VALUE_INFO_NAME       0x02
#define MSK_SIII_SL_IDN_CREATE_IDN_VALUE_INFO_UNIT       0x04
#define MSK_SIII_SL_IDN_CREATE_IDN_VALUE_INFO_MINIMUM    0x08
#define MSK_SIII_SL_IDN_CREATE_IDN_VALUE_INFO_MAXIMUM    0x10
#define MSK_SIII_SL_IDN_CREATE_IDN_VALUE_INFO_VALUE      0x20

typedef struct SIII_SL_IDN_CREATE_IDN_REQ_Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  SIII_SL_IDN_CREATE_IDN_REQ_DATA_T   tData;
} SIII_SL_IDN_CREATE_IDN_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_CREATE_IDN_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	20+n	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A40	SIII_SL_IDN_CMD_CREATE_IDN_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_CREATE_IDN_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
bValueInfo	UINT8	Bit field	Value info (which data elements to create)
usMaxNameLength	UINT16	0..240	Maximum length of name. Set to 0 to prevent IDN from having the element “Name”.
usMaxUnitLength	UINT16	0..48	Maximum length of unit. Set to 0 to prevent IDN from having the element “Unit”.
usMaxListDataSize	UINT16		Maximum list data size specified as number of bytes (byte length includes the list header). It is only valid on List types
ulAttribute	UINT32		Attribute
ulTotalLength	UINT32		Total Length of data
abData[1]	UINT8[]		Data block of IDN

Table 30: SIII_SL_IDN_CREATE_IDN_REQ_T – Create IDN Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_CREATE_IDN_CNF_DATA_Ttag
{
  TLR_UINT32                          ulIdn;
  TLR_UINT8                           bSlaveIdx;
} SIII_SL_IDN_CREATE_IDN_CNF_DATA_T;

typedef struct SIII_SL_IDN_CREATE_IDN_CNF_Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  SIII_SL_IDN_CREATE_IDN_CNF_DATA_T  tData;
} SIII_SL_IDN_CREATE_IDN_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_CREATE_IDN_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A41	SIII_SL_IDN_CMD_CREATE_IDN_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_CREATE_IDN_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 31: SIII_SL_IDN_CREATE_IDN_CNF_T – Create IDN Confirmation Packet

5.4.2 Register for IDN read or write access

Note: It is possible to register the created IDNs listed in chapter “IDNs created by the Stack”at the stack. These IDNs will be created if the stack is configured by SIII_SL_AP_SET_CONFIGURATION_REQ followed by a RCX_CHANNEL_INIT_REQ packet.

If the master reads or writes the operation data of an IDN, it is possible to register for receiving a notification.

Three cases can be considered:

- Use the read notification service on operation data if changing data from the application shall get mapped into an IDN.
- Use the write notification service on operation data if the application want’s to receive data from the master.
- Use the read notification service on the attribute if the attribute shall get changed during runtime.

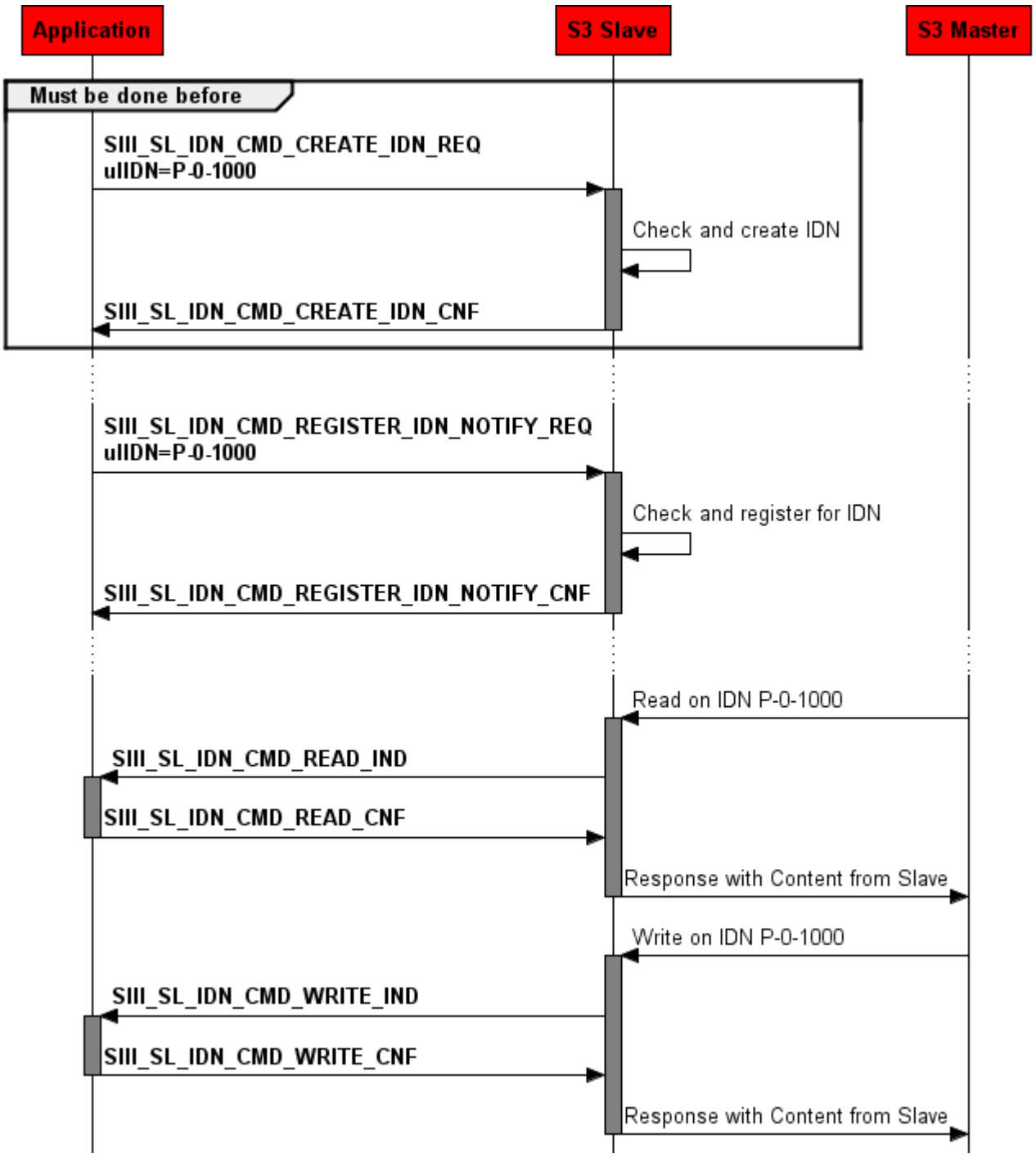


Figure 34: Create IDN and register read and write notification

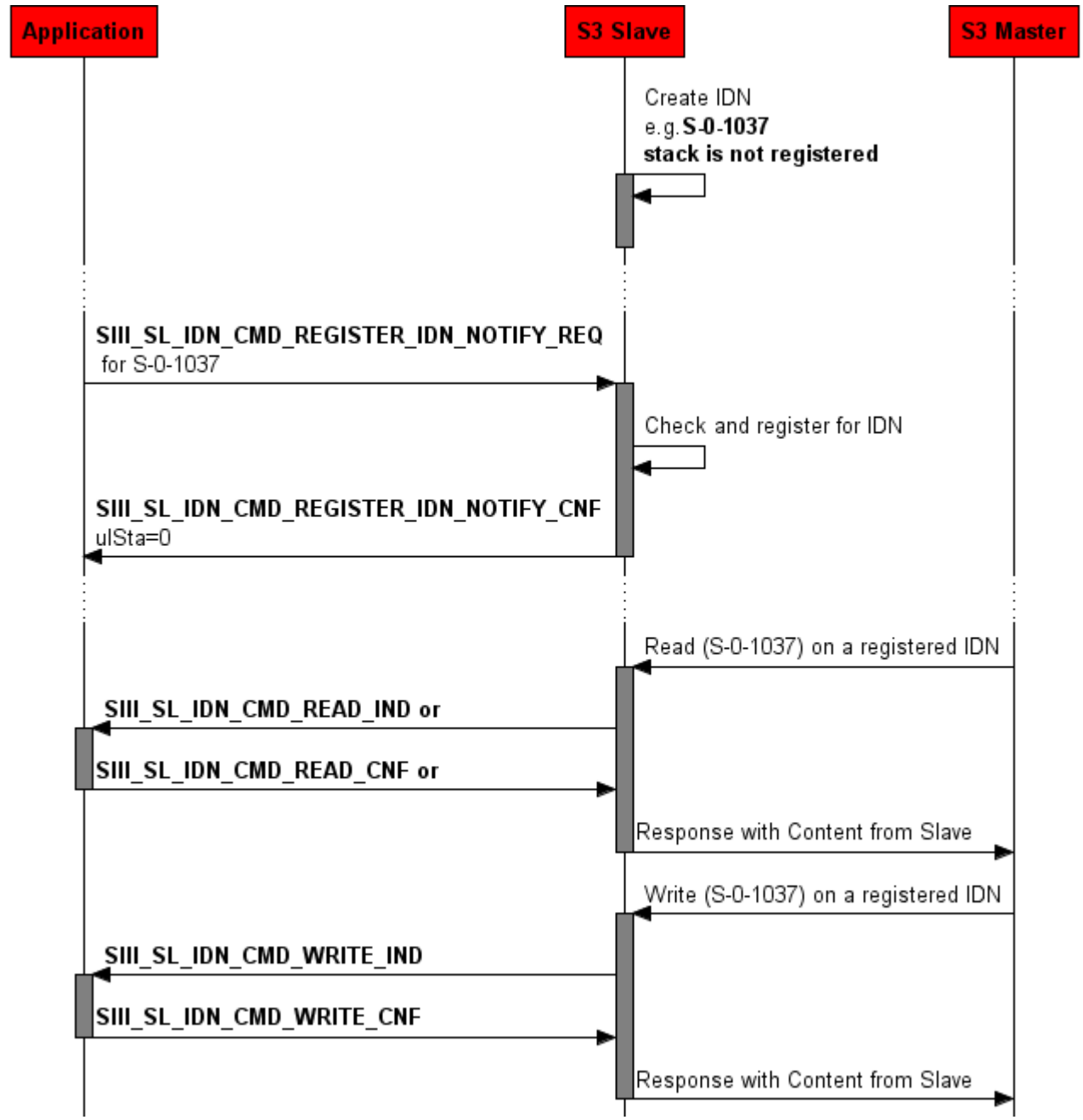


Figure 35: Register for stack created IDN

Note: It is not possible to register a notification for all IDNs. Some of them already have been registered internally by the stack. The user cannot register to these IDNs.

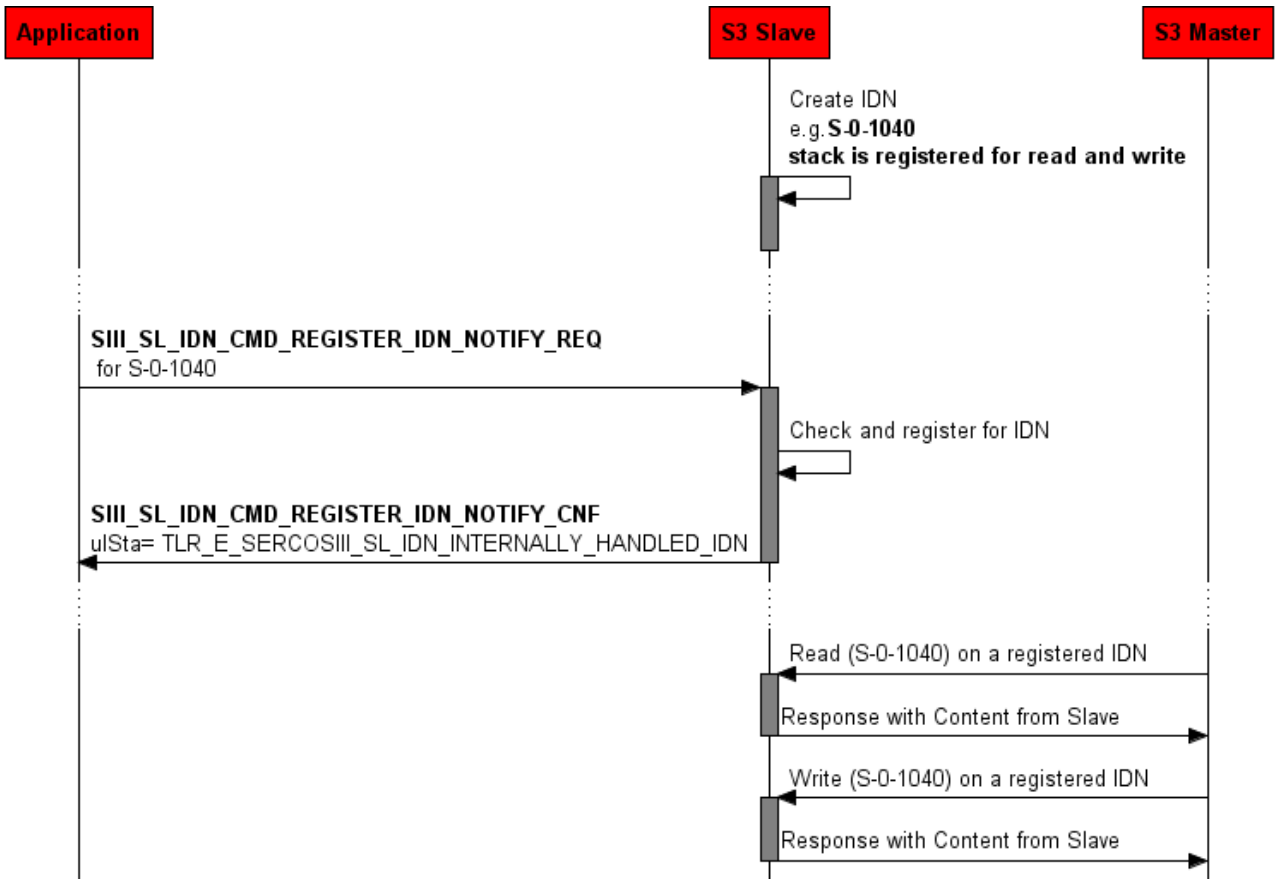


Figure 36: Register for already registered IDN

A registered IDN must be answered within 5 seconds. If not, the stack will generate an error.

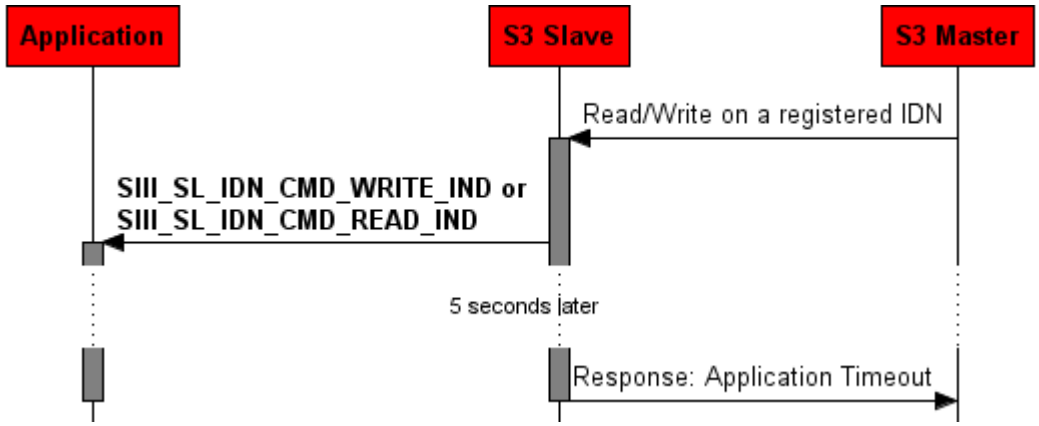


Figure 37: Timeout of registered IDN

Packet Structure Reference

```
typedef struct SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT32                                ulIdn;
  TLR_UINT8                                 bSlaveIdx;
  TLR_BOOLEAN8                              fReadNotify;
  TLR_BOOLEAN8                              fWriteNotify;
  TLR_BOOLEAN8                              fVirtualMode;
  TLR_BOOLEAN8                              fAttributeReadNotify;
} SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_DATA_T;

typedef struct SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_DATA_T  tData;
} SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	9	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A44	SIII_SL_IDN_CMD_REGISTER_IDN_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0…7	Index of the slave device
fReadNotify	BOOLEAN8	0,1	Read notify on operation data
fWriteNotify	BOOLEAN8	0,1	Write notify on operation data
fVirtualMode	BOOLEAN8	0,1	Virtual mode. Just use the default value 0.
fAttributeReadNotify	BOOLEAN8	0,1	This read notify attribute enables the application to provide changing Write Protection flags that way

Table 32: SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_T – Register IDN Notify Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT32                                ulIdn;
  TLR_UINT8                                 bSlaveIdx;
} SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_DATA_T;

typedef struct SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_DATA_T  tData;
} SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A45	SIII_SL_IDN_CMD_REGISTER_IDN_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0…7	Index of the slave device

Table 33: `SIII_SL_IDN_REGISTER_IDN_NOTIFY_CNF_T` – Register IDN Notify Confirmation Packet

5.4.3 Write IDN Indication

The packet `SIII_SL_IDN_CMD_WRITE_REQ/IND` is used for write access to the object dictionary. Up to 65536 bytes of data can be stored into a single IDN within the object dictionary.

This packet can be used in two manners with the same command code: as request (direction host → device) and also as indication (direction device → host).

It is sent as request if the IDN is located on the netX and the user wants to modify it. In this case, all elements except operation data are read only and cannot be changed via the Write IDN Request packet. However, there are separate packets to change the IDN name, unit, data status, minimum and maximum values (see chapters 5.5.2 to 5.5.5). Attribute is always write-protected. So for the IDNs located on the netX (standard or self-created) only element operation data can be changed via Write IDN Request packet.

It is sent as indication if the IDN is managed by the user and located on the host or in the user application (undefined IDN) and the master or communication stack wants to modify it. For these IDNs the elements name, unit, minimum, maximum values and operation data can be changed via Write IDN Request/Indication packet. The element's attribute and data status are read only. The packets described in chapters 5.5.2 to 5.5.5 are not suitable for undefined IDNs (no write IDN indication sent to the application).

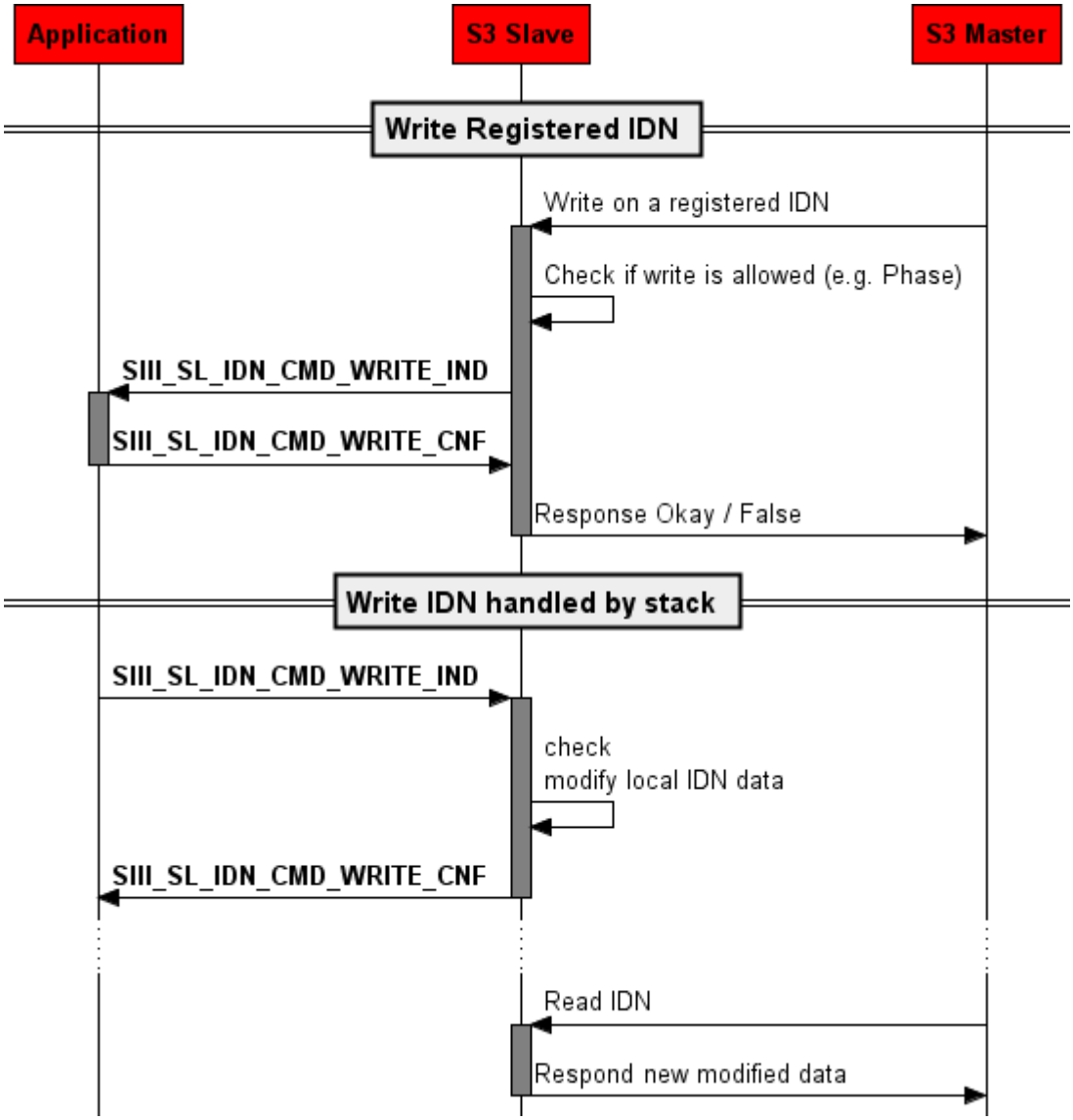


Figure 38: Sequence Diagram for the SIII_SL_IDN_CMD_WRITE_IND/CNF Packet

The parameter bDataBlockElement decides which part of the IDN is accessed. The following values may be chosen for this parameter:

Value	Name	Description
1	SIII_SL_IDN_ELEMENT_DATASTATUS	Access the data status.
2	SIII_SL_IDN_ELEMENT_NAME	Access the name of the IDN.
3	SIII_SL_IDN_ELEMENT_ATTRIBUTE	Access the attribute of the IDN.
4	SIII_SL_IDN_ELEMENT_UNIT	Access the unit of the IDN.
5	SIII_SL_IDN_ELEMENT_MINIMUM_VALUE	Access the minimum of the IDN.
6	SIII_SL_IDN_ELEMENT_MAXIMUM_VALUE	Access the maximum of the IDN.
7	SIII_SL_IDN_ELEMENT_OPDATA	Access the operation data of the IDN.

Table 34: Possible Values for Parameter bDataBlockElement

The parameter array abData[] may contain either a single value (scalar) or a list. In case of a scalar, only the value is contained. Contrarily, in case of a list, the parameter array abData[] contains the data preceded by a write list header.

The use of the write list header is shown by the following example:

```
tSendPacket.tHeader.ulCmd = SIII_SL_IDN_CMD_WRITE_REQ;
tSendPacket.tHeader.ulLen = SIII_SL_IDN_WRITE_REQ_MIN_SIZE + 12;
tSendPacket.tHeader.ulState = 0;
tSendPacket.tHeader.ulExt = 0;
ptPck->tData.bSlaveIdx = bSlaveIdx;
ptPck->tData.ulIdn = SIII_SL_IDN(S, 0, 14, 0, 0);
ptPck->tData.bDataBlockElement = SIII_SL_IDN_ELEMENT_OPDATA;
ptPck->tData.usTotalLength = 8+4;

/* write list header */
{
UINT16 usCurListLen = 8; /* current list length */
UINT16 usMaxListLen = 0; /* max. list len */
memcpy (&ptPck->tData.abData[0],&usCurListLen,2);
memcpy (&ptPck->tData.abData[2],&usMaxListLen,2);
}
memcpy(&ptPck->tData.abData[4], "NEW_NAME", 8);
```

Packet Structure Reference

```
#define SIII_SL_IDN_WRITE_REQ_MIN_SIZE ((unsigned
long)(((SIII_SL_IDN_WRITE_REQ_DATA_T*)0)->abData))

typedef struct SIII_SL_IDN_WRITE_REQ_DATA_Ttag
{
  TLR_UINT32  ulIdn;
  TLR_UINT8   bSlaveIdx;
  TLR_UINT8   bDataBlockElement;
  TLR_UINT16  usTotalLength;
  TLR_UINT8   abData[SIII_SL_IDN_NUM_READWRITE_DATA_BYTES];
} SIII_SL_IDN_WRITE_REQ_DATA_T;

typedef struct SIII_SL_IDN_WRITE_REQ_Ttag
{
  TLR_PACKET_HEADER_T                    tHead;
  SIII_SL_IDN_WRITE_REQ_DATA_T           tData;
} SIII_SL_IDN_WRITE_REQ_T;

/* indication packet */
#define SIII_SL_IDN_WRITE_IND_MIN_SIZE ((unsigned long)(((SIII_SL_IDN_WRITE_IND_DATA_T*)0)->abData))

typedef SIII_SL_IDN_WRITE_REQ_DATA_T   SIII_SL_IDN_WRITE_IND_DATA_T;
typedef SIII_SL_IDN_WRITE_REQ_T        SIII_SL_IDN_WRITE_IND_T;
```

Packet Description

Structure SIII_SL_IDN_WRITE_REQ/IND_T			Type: Request/Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	8+n	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	0	See section *Status/Error Codes Overview* Note: ulSta must always be zero in SIII_SL_IDN_CMD_WRITE_REQ/IND otherwise the transfer is aborted
ulCmd	UINT32	0x5A00	SIII_SL_IDN_CMD_WRITE_REQ/IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_WRITE_REQ_DATA_T/SIII_SL_IDN_WRITE_IND_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
bDataBlockElement	UINT8	1..7	Data Block Element to be accessed
usTotalLength	UINT16	1..1024	Total Length of data to be written
abData[1024]	UINT8[]		Data to be written (field)

Table 35: `SIII_SL_IDN_WRITE_REQ/IND_T` – Write Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_WRITE_CNF_DATA_Ttag
{
  TLR_UINT32  ulIdn;
  TLR_UINT8   bSlaveIdx;
  TLR_UINT8   bDataBlockElement;
} SIII_SL_IDN_WRITE_CNF_DATA_T;

typedef struct SIII_SL_IDN_WRITE_CNF_Ttag
{
  TLR_PACKET_HEADER_T           tHead;
  SIII_SL_IDN_WRITE_CNF_DATA_T  tData;
} SIII_SL_IDN_WRITE_CNF_T;

typedef SIII_SL_IDN_WRITE_CNF_T SIII_SL_IDN_WRITE_RES_T;
```

Packet Description

Structure SIII_SL_IDN_WRITE_CNF/RES_T			**Type: Confirmation/Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	6	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A01	SIII_SL_IDN_CMD_WRITE_CNF/RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_WRITE_CNF/RES_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
bDataBlockElement	UINT8	1..7	Data Block Element having been accessed

Table 36: SIII_SL_IDN_WRITE_CNF/RES_T – Write Confirmation /Response Packet

5.4.4 Read IDN Indication

The packet `SIII_SL_IDN_CMD_READ_REQ/IND/CNF/RES` is used for read access to the object dictionary where the IDNs are stored into. Up to 65536 bytes of data can be retrieved from a single IDN within the object dictionary.

This packet can be used as request (direction host → device) and also with the same command code as indication (direction device → host). It is send as indication if the IDN is located on the host or in the user application and the master or communication stack want to read it. It is send as request if the IDN is located on the netX and the user wants to read it.

Note: If IDNs get just registered by the packet `SIII_SL_IDN_REGISTER_IDN_NOTIFY_REQ_T` the stack will just send read notifications for operational data. If the registration is done by the packet `SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_T` all notifications for an IDN shall be implemented and handled (e.g. Name, Minimum/Maximum Value, etc.).

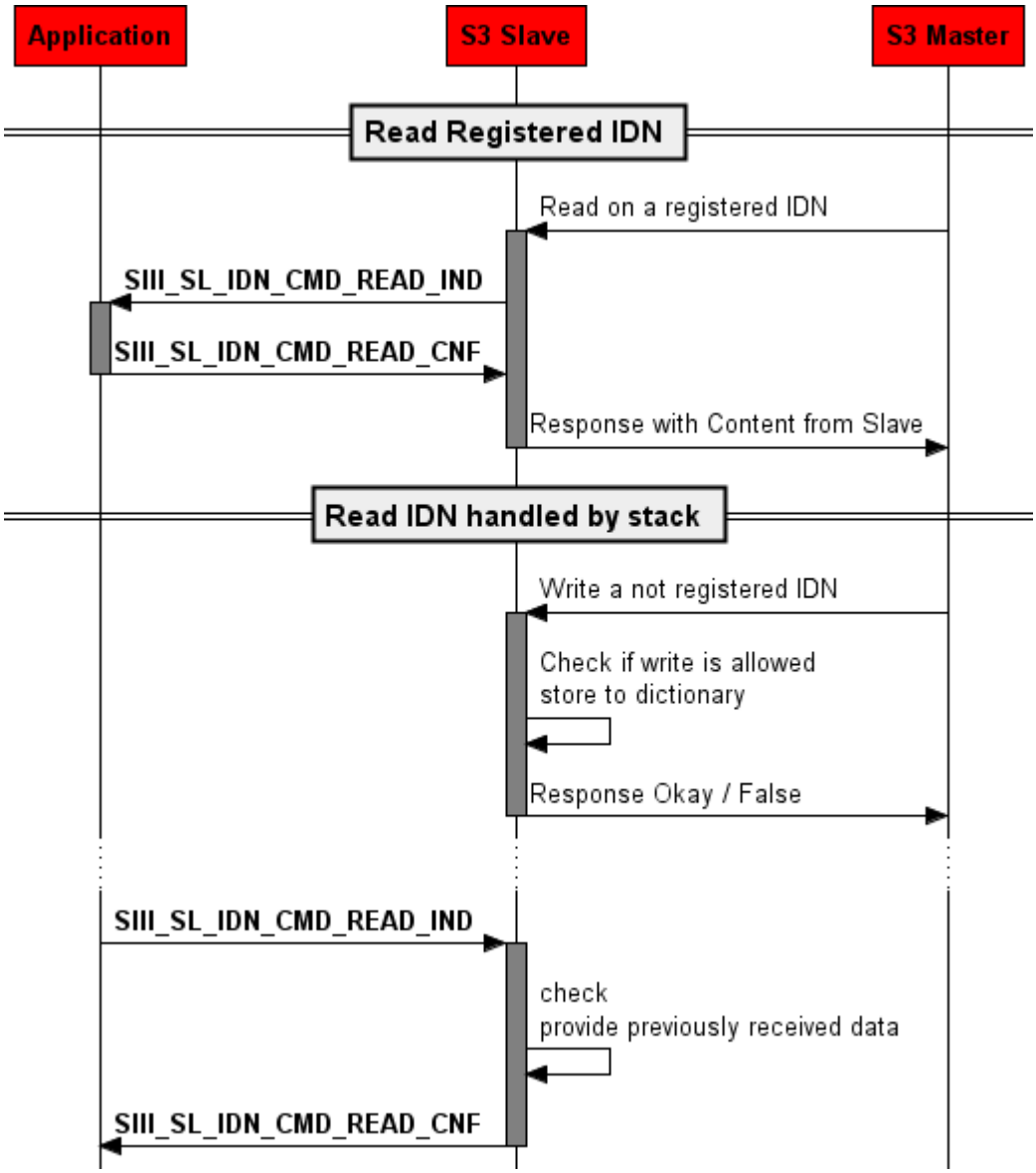


Figure 39: Sequence Diagram for the SIII_SL_IDN_CD_READ_IND/CNF Packet

The parameter `bDataBlockElement` decides which part of the IDN is accessed. The following values may be chosen for this parameter:

Value	Name	Description
1	SIII_SL_IDN_ELEMENT_DATASTATUS	Access the data status of the IDN.
2	SIII_SL_IDN_ELEMENT_NAME	Access the name of the IDN.
3	SIII_SL_IDN_ELEMENT_ATTRIBUTE	Access the attribute of the IDN.
4	SIII_SL_IDN_ELEMENT_UNIT	Access the unit of the IDN.
5	SIII_SL_IDN_ELEMENT_MINIMUM_VALUE	Access the minimum of the IDN.
6	SIII_SL_IDN_ELEMENT_MAXIMUM_VALUE	Access the maximum of the IDN.
7	SIII_SL_IDN_ELEMENT_OPDATA	Access the operation data of the IDN.

Table 37: Possible values for Parameter bDataBlockElement

Packet Structure Reference

```
typedef struct SIII_SL_IDN_READ_REQ_DATA_Ttag
{
  TLR_UINT32  ulIdn;
  TLR_UINT8   bSlaveIdx;
  TLR_UINT8   bDataBlockElement;
} SIII_SL_IDN_READ_REQ_DATA_T;

typedef struct SIII_SL_IDN_READ_REQ_Ttag
{
  TLR_PACKET_HEADER_T          tHead;
  SIII_SL_IDN_READ_REQ_DATA_T  tData;
} SIII_SL_IDN_READ_REQ_T;

/* indication packet */
typedef SIII_SL_IDN_READ_REQ_DATA_T  SIII_SL_IDN_READ_IND_DATA_T;
typedef SIII_SL_IDN_READ_REQ_T       SIII_SL_IDN_READ_IND_T;
```

Packet Description

Structure SIII_SL_IDN_READ_REQ/IND_T			Type: Request/Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	6	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	0	See section *Status/Error Codes Overview* Note: ulSta must always be zero in SIII_SL_IDN_CMD_READ_REQ/IND otherwise the transfer is aborted
ulCmd	UINT32	0x5A02	SIII_SL_IDN_CMD_READ_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_READ_REQ/IND_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
bDataBlockElement	UINT8	1..7	Data Block Element to be accessed (see *Table 37: Possible values for Parameter `bDataBlockElement`*)

Table 38: `SIII_SL_IDN_READ_REQ/IND_T` – Read Request Packet

Packet Structure Reference

```
#define SIII_SL_IDN_READ_CNF_MIN_SIZE
                          ((unsigned long)(((SIII_SL_IDN_READ_CNF_DATA_T*)0)->abData))

typedef struct SIII_SL_IDN_READ_CNF_DATA_Ttag
{
  TLR_UINT32  ulIdn;
  TLR_UINT8   bSlaveIdx;
  TLR_UINT8   bDataBlockElement;
  TLR_UINT16  usTotalLength;
  TLR_UINT8   abData[SIII_SL_IDN_NUM_READWRITE_DATA_BYTES];
} SIII_SL_IDN_READ_CNF_DATA_T;

typedef struct SIII_SL_IDN_READ_CNF_Ttag
{
  TLR_PACKET_HEADER_T          tHead;
  SIII_SL_IDN_READ_CNF_DATA_T  tData;
} SIII_SL_IDN_READ_CNF_T;

#define SIII_SL_IDN_READ_RES_MIN_SIZE
                          ((unsigned long)(((SIII_SL_IDN_READ_RES_DATA_T*)0)->abData))

typedef SIII_SL_IDN_READ_CNF_DATA_T SIII_SL_IDN_READ_RES_DATA_T;
typedef SIII_SL_IDN_READ_CNF_T SIII_SL_IDN_READ_RES_T;
```

Packet Description

Structure SIII_SL_IDN_READ_CNF/RES_T			**Type: Confirmation/Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A03	SIII_SL_IDN_CMD_READ_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_READ_CNF/RES_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
bDataBlockElement	UINT8	1..7	Data Block Element to be accessed (see *Table 37: Possible values for Parameter bDataBlockElement*)
usTotalLength	UINT16	1..1024	Total Length of data having been read
abData[1024]	UINT8		Data having been read (field)

Table 39: SIII_SL_IDN_READ_CNF_T – Read IDN Confirmation Pac

5.5 Additional IDN Handling Packets

The following packets (5.5.1 to 5.5.5) are only need in some rare special cases.

5.5.1 Change minimum and maximum values of stack internal IDN

This packet shall be used if the slave does not support all Sercos specified cycle times. Dependent on the amount of IOData, CPU speed and application size not all cycle times are supported. In this case the minimum and maximum values of the IDNs

- S-0-1002 (t_{Scyc} – Communication cycle time)
- S-0-1050.x.10 (Producer Cycle time)

shall be changed to the supported values. In normal cases just the minimum cycle time must be changed, the maximum cycle time shall again be set to 65 ms. Use as minimum cycle time 250 µs which currently is the minimum of the most common masters. Cycle times below 250 µs require special tests and investigation whether the device will work properly in all situations.

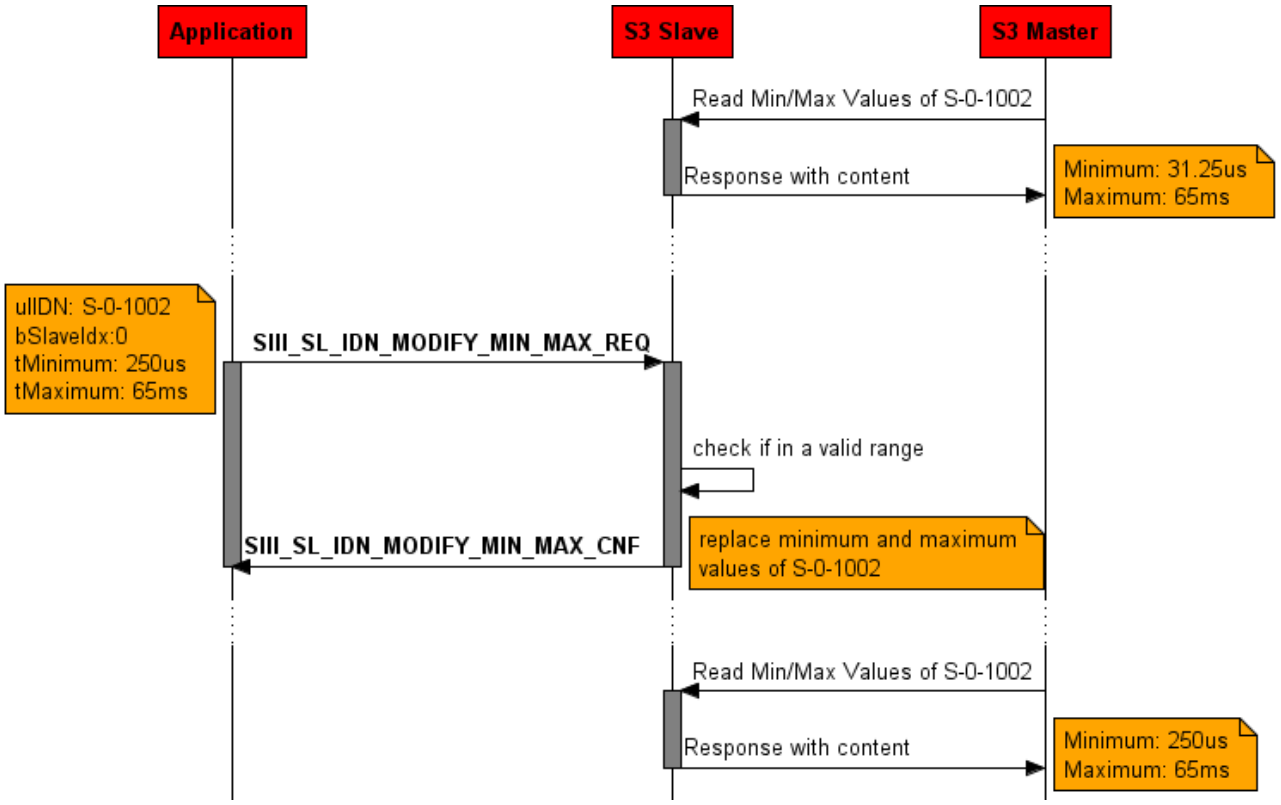


Figure 40: Change minimum and maximum values of stack internal IDN

Note: This packet can used for IDNs still created in the stack. If an IDN has no minimum or maximum no value can get assigned or changed.

Packet Structure Reference

```
typedef union SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_VALUE_Ttag
{
  TLR_UINT8                                        bValue;
  TLR_UINT16                                       usValue;
  TLR_UINT32                                       ulValue;
  TLR_UINT64                                       ull64BitValue;
  TLR_UINT8                                        abData[8];
} SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_VALUE_T;

typedef struct SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_Ttag
{
  TLR_UINT32                                       ulIdn;
  TLR_UINT8                                        bSlaveIdx;
  SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_VALUE_T      tMinimum;
  SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_VALUE_T      tMaximum;
} SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_T;

typedef struct SIII_SL_IDN_MODIFY_MIN_MAX_REQ_Ttag
{
  TLR_PACKET_HEADER_T                              tHead;
  SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_T            tData;
} SIII_SL_IDN_MODIFY_MIN_MAX_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_MODIFY_MIN_MAX_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A54	SIII_SL_IDN_CMD_MODIFY_MIN_MAX_REQ- Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_MODIFY_MIN_MAX_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
tMinimum	Union	0 ... 2^{32}-1	Choose the right variable from the union. The variable respectively the size is limited by the IDN (see Sercos Spec). For Example see IDN-0-1040 the data length is 2 octets. So write your new minimum value to `tMinimum.usValue`.
tMaximum	Union	0 ... 2^{32}-1	See `tMinimum`.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_MODIFY_MIN_MAX_CNF_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_MODIFY_MIN_MAX_CNF_DATA_T;

typedef struct SIII_SL_IDN_MODIFY_MIN_MAX_CNF_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_MODIFY_MIN_MAX_CNF_DATA_T         tData;
} SIII_SL_IDN_MODIFY_MIN_MAX_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_CMD_MODIFY_MIN_MAX_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A55	SIII_SL_IDN_CMD_MODIFY_MIN_MAX_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_MODIFY_MIN_MAX_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 40: SIII_SL_IDN_CMD_MODIFY_MIN_MAX_CNF_T – Change minimum and maximum values of internal IDN Confirmation Packet

5.5.2 Set IDN Name Request

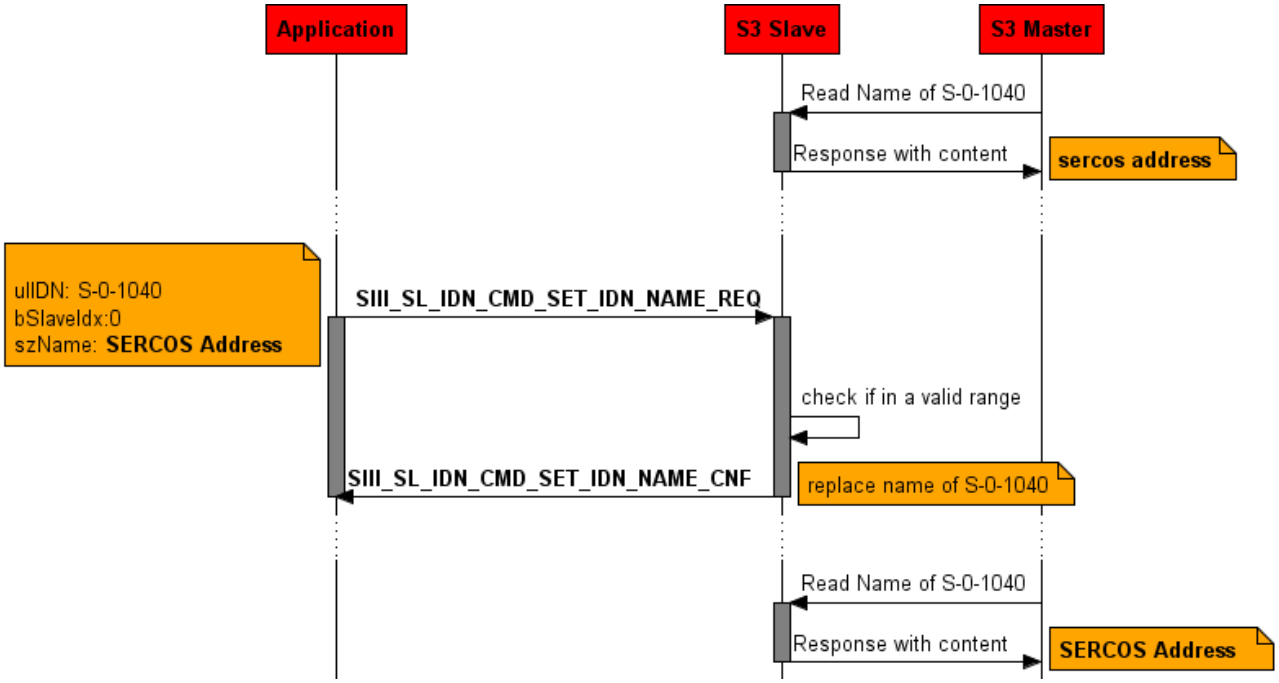


Figure 41: Set IDN name request

This packet can be used to change the interface language during runtime (see IDN S-0-0265 Language selection).

Note: This packet is used for IDNs still created in the stack. Ordinary devices will not need this packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_IDN_NAME_REQ_DATA_Ttag
{
  TLR_UINT32                          ulIdn;
  TLR_UINT8                           bSlaveIdx;
  TLR_UINT8                           szName[240];
} SIII_SL_IDN_SET_IDN_NAME_REQ_DATA_T;

typedef struct SIII_SL_IDN_SET_IDN_NAME_REQ_Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  SIII_SL_IDN_SET_IDN_NAME_REQ_DATA_T tData;
} SIII_SL_IDN_SET_IDN_NAME_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_SET_IDN_NAME_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5..245	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A4C	SIII_SL_IDN_CMD_SET_IDN_NAME_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_IDN_NAME_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
szName[240]	UINT8[]		This field contains the new name to be set. It may be up to 240 bytes long.

Table 41: SIII_SL_IDN_SET_IDN_NAME_REQ_DATA_T – Set IDN Name Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_IDN_NAME_CNF_DATA_Ttag
{
  TLR_UINT32                          ulIdn;
  TLR_UINT8                           bSlaveIdx;
} SIII_SL_IDN_SET_IDN_NAME_CNF_DATA_T;

typedef struct SIII_SL_IDN_SET_IDN_NAME_CNF_Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  SIII_SL_IDN_SET_IDN_NAME_CNF_DATA_T tData;
} SIII_SL_IDN_SET_IDN_NAME_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_SET_IDN_NAME_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A4D	SIII_SL_IDN_CMD_SET_IDN_NAME_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_IDN_NAME_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 42: SIII_SL_IDN_SET_IDN_NAME_CNF_DATA_T – Set IDN Name Confirmation Packet

5.5.3 Change unit of stack internal IDN

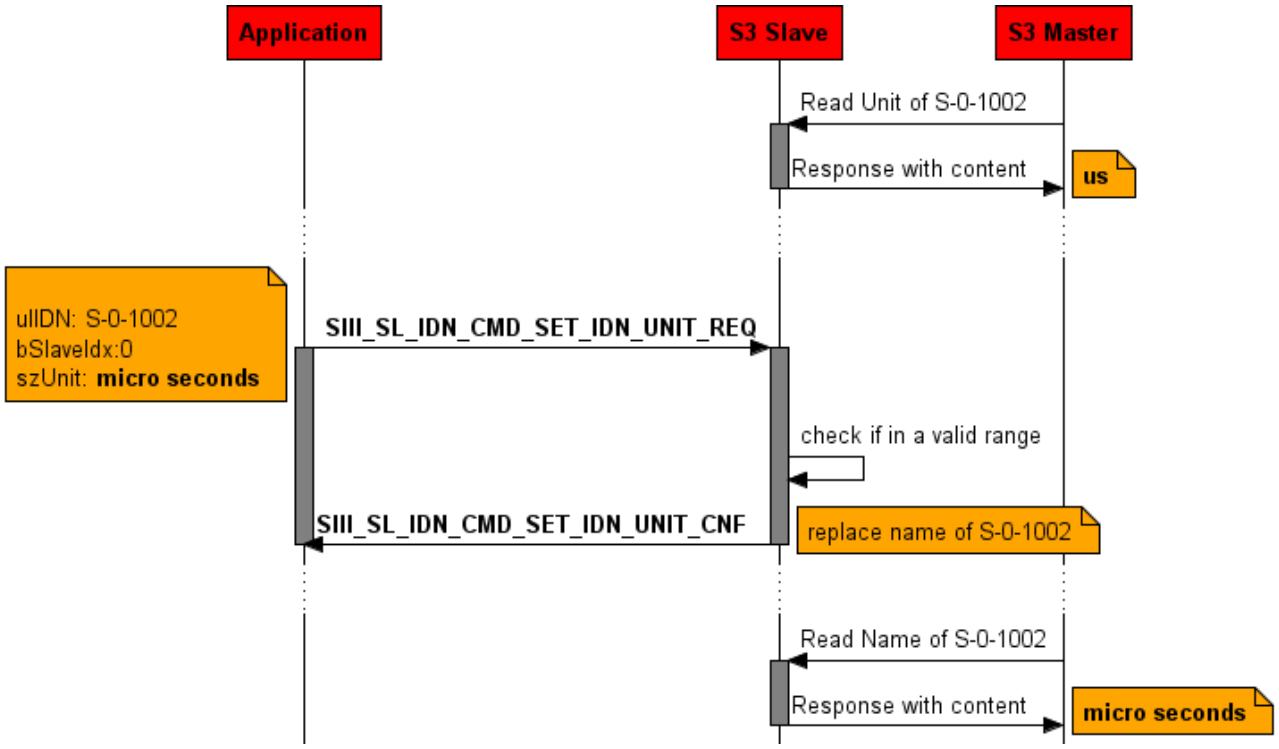


Figure 42: Change unit of stack internal IDN

This packet can be used to change the interface language during runtime (see IDN S-0-0265 Language selection).

Note: This packet can just be used for IDNs still created in the stack with a unit. Ordinary devices will not need this packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_IDN_UNIT_REQ_DATA_Ttag
{
  TLR_UINT32                          ulIdn;
  TLR_UINT8                           bSlaveIdx;
  TLR_UINT8                           szUnit[48];
} SIII_SL_IDN_SET_IDN_UNIT_REQ_DATA_T;

#define SIII_SL_IDN_SET_IDN_UNIT_REQ_MIN_SIZE  (5) /* the value starts at szUnit */

typedef struct SIII_SL_IDN_SET_IDN_UNIT_REQ_Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  SIII_SL_IDN_SET_IDN_UNIT_REQ_DATA_T  tData;
} SIII_SL_IDN_SET_IDN_UNIT_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_SET_IDN_UNIT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5..53	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A4E	SIII_SL_IDN_CMD_SET_IDN_UNIT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_IDN_UNIT_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device
szUnit[48]	UINT8[]		This field contains the new unit name to be set. It may be up to 48 bytes long.

Table 43: `SIII_SL_IDN_SET_IDN_UNIT_REQ_T` – Set IDN Unit Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_IDN_UNIT_CNF_DATA_Ttag
{
  TLR_UINT32                          ulIdn;
  TLR_UINT8                           bSlaveIdx;
} SIII_SL_IDN_SET_IDN_UNIT_CNF_DATA_T;

typedef struct SIII_SL_IDN_SET_IDN_UNIT_CNF_Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  SIII_SL_IDN_SET_IDN_UNIT_CNF_DATA_T  tData;
} SIII_SL_IDN_SET_IDN_UNIT_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_SET_IDN_UNIT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A4F	SIII_SL_IDN_CMD_SET_IDN_UNIT_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_IDN_UNIT_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 44: SIII_SL_IDN_SET_IDN_UNIT_CNF_T – Set IDN Unit Confirmation Packet

5.5.4 Set data status of the IDN to valid

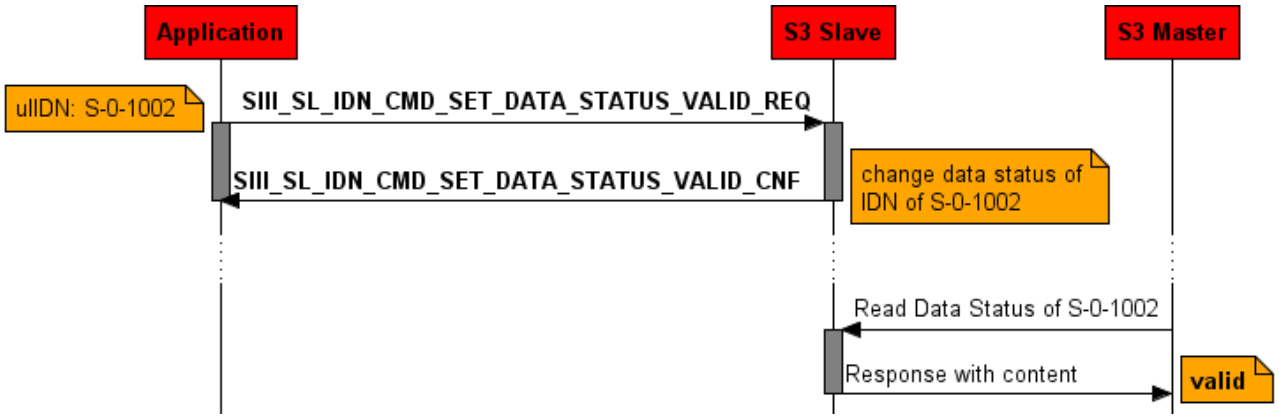


Figure 43: Set Data status of IDN to valid

The packet `SIII_SL_IDN_CMD_SET_DATA_STATUS_VALID_REQ/CNF` allows setting the data status of the IDN specified in parameter `ulIdn` to valid. This packet affects only IDNs which are no procedure commands.

The data status signals whether the data itself is valid or not.

The data status of an IDN can be preset when the IDN is created (`SIII_SL_IDN_CMD_CREATE_IDN_REQ`). It can be set to valid using this command. It can be set to invalid by the command (`SIII_SL_IDN_CMD_SET_DATA_STATUS_INVALID_REQ`). When the master writes an IDN, the data status is automatically set to valid.

Note: This packet shall be used for IDNs still created in the stack with a unit. Ordinary devices will not need this packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_DATA_T;

typedef struct SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_DATA_T  tData;
} SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_ID N	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A50	SIII_SL_IDN_CMD_SET_DATA_STATUS_VALID_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 45: `SIII_SL_IDN_SET_DATA_STATUS_VALID_REQ_T` – Set Data State Valid Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_DATA_T;

typedef struct SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_DATA_T  tData;
} SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A51	SIII_SL_IDN_CMD_SET_DATA_STATUS_VALID_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 46: SIII_SL_IDN_SET_DATA_STATUS_VALID_CNF_T – Set Data State Valid Confirmation Packet

5.5.5 Set data status of the IDN to invalid

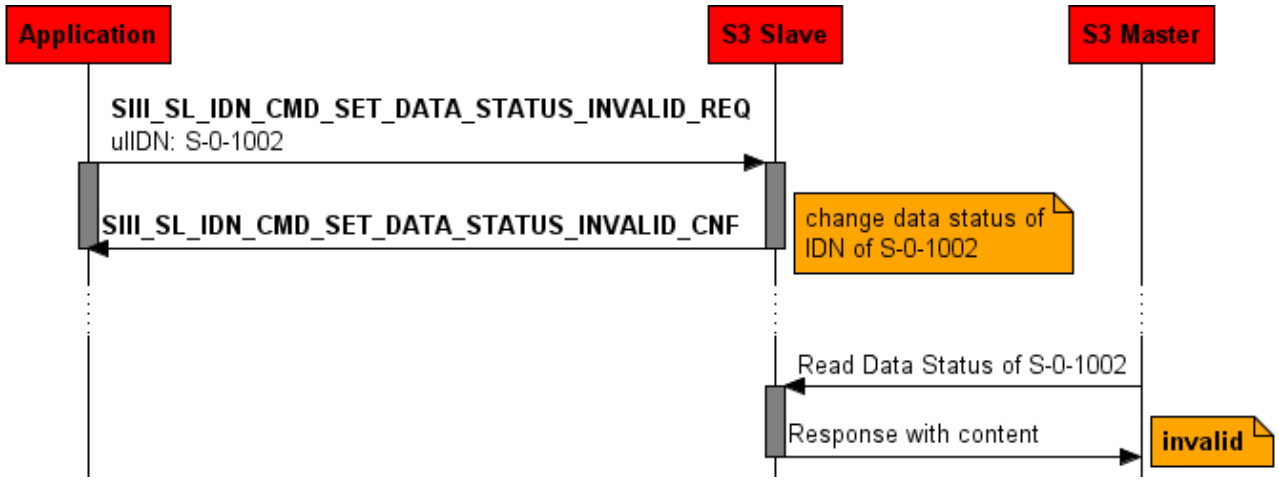


Figure 44: Set Data Status of the IDN to invalid

The packet `SIII_SL_IDN_CMD_SET_DATA_STATUS_INVALID_REQ/CNF` allows you to set the data status of the IDN specified in parameter `ulIdn` to invalid.

This packet affects only IDNs which are no commands.

Note: This packet shall be used for IDNs still created in the stack with a unit. Ordinary devices will not need this packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_DATA_T;

typedef struct SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_DATA_T  tData;
} SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A52	SIII_SL_IDN_CMD_SET_DATA_STATUS_INVALID_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 47: `SIII_SL_IDN_SET_DATA_STATUS_INVALID_REQ_T` – Set Data Status Invalid Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_DATA_T;

typedef struct SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_DATA_T  tData;
} SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A53	SIII_SL_IDN_CMD_SET_DATA_STATUS_INVALID_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the Sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 48: `SIII_SL_IDN_SET_DATA_STATUS_INVALID_CNF_T` – Set Data Status Invalid Confirmation Packet

5.5.6 Register Undefined Notify Request

The packet `SIII_SL_IDN_CMD_REGISTER_UNDEFINED_NOTIFY_REQ/CNF` is used for registering a notification occurring on access to an undefined IDN. This is used, if the user manages IDNs on his own. The stack does not know those and always asks the user when the master accesses an undefined IDN.

Consequences of this packet:

If the S-0-17 (IDN-list of all operation data) or S-0-25 (IDN-list of all procedure commands) is read by the master, the stack asks the user for a list of the managed IDNs. The user managed IDNs and stack managed IDNs are merged and provided to the master.

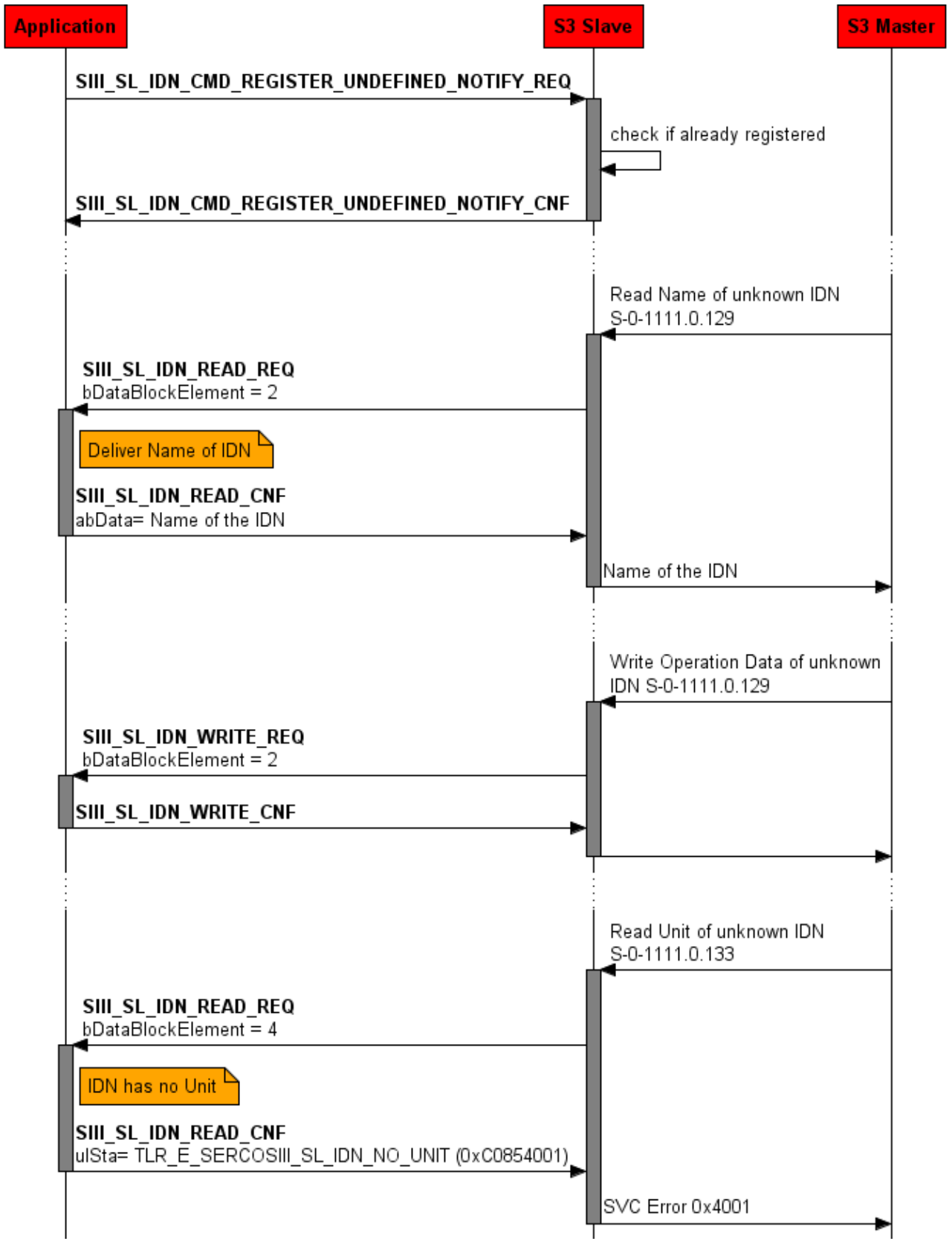


Figure 45: SIII_SL_IDN_CMD_REGISTER_UNDEFINED_NOTIFY_REQ

Note: Just use this packet if a large amount IDNs shall be handled in the application. Otherwise, it is better to use the normal IDN handling procedure (see preceding chapters).

Packet Structure Reference

```
typedef struct SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_DATA_T;

typedef struct SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_DATA_T   tData;
} SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A48	SIII_SL_IDN_CMD_REGISTER_UNDEFINED_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 49: SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_REQ_T – Register Undefined Notify Request Packet

5.5.7 Register Undefined Notify Confirmation

This packet is returned from the stack to the application after the stack has processed the Register Undefined Notify Request packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_DATA_T;

typedef struct SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_DATA_T   tData;
} SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A49	SIII_SL_IDN_CMD_REGISTER_UNDEFINED_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 50: SIII_SL_IDN_REGISTER_UNDEFINED_NOTIFY_CNF_T – Register Undefined Notify Confirmation Packet

5.5.8 Unregister Undefined Notify Request

The packet SIII_SL_IDN_CMD_UNREGISTER_UNDEFINED_NOTIFY_REQ/CNF is used for unregistering a notification occurring on read or write access of an IDN which has previously been established using the SIII_SL_IDN_CMD_REGISTER_UNDEFINED_NOTIFY_REQ packet.

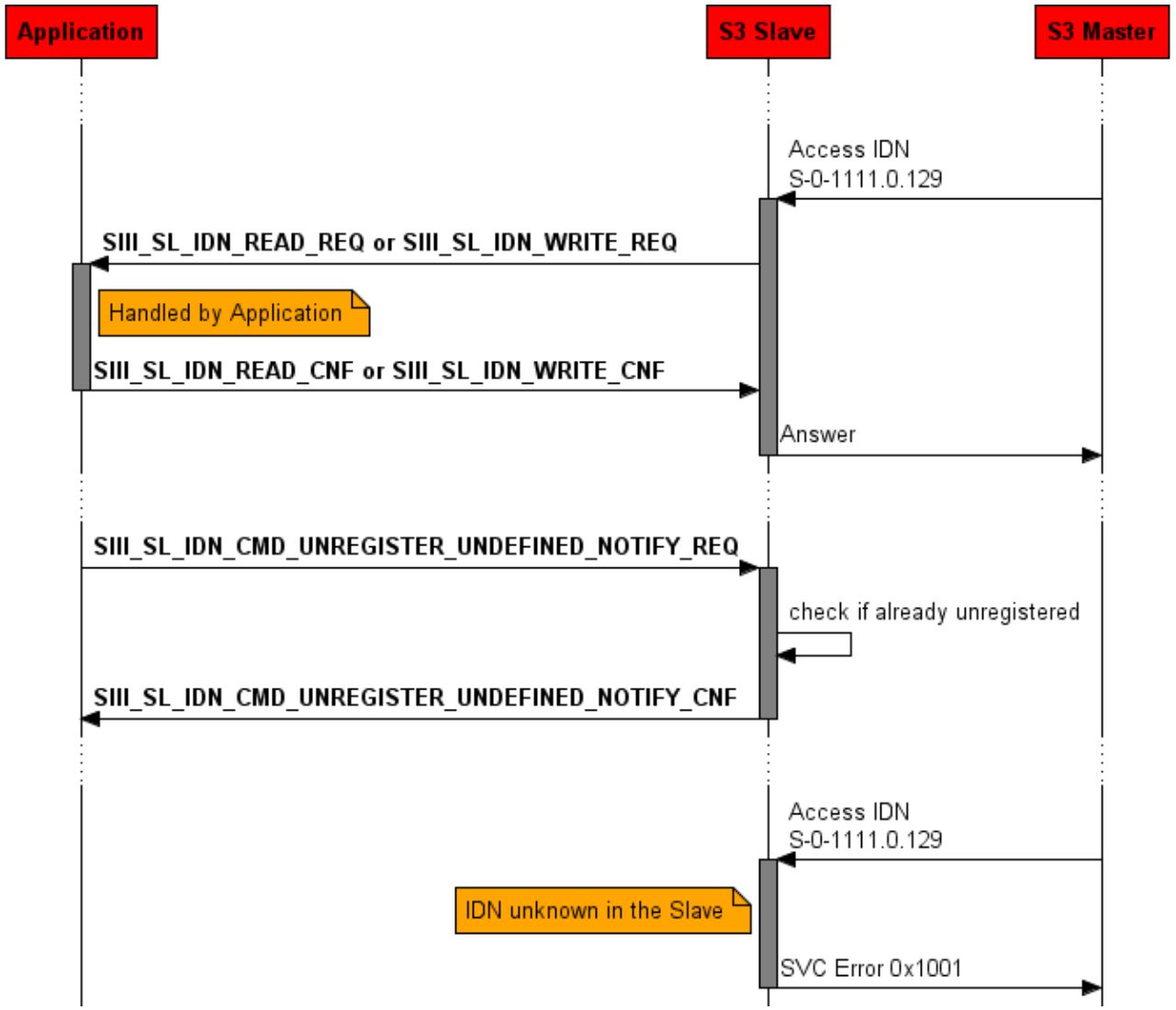


Figure 46: SIII_SL_IDN_CMD_UNREGISTER_UNDEFINED_NOTIFY_REQ

Packet Structure Reference

```
typedef struct SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_DATA_T;

typedef struct SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_DATA_T tData;
} SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A4A	SIII_SL_IDN_CMD_UNREGISTER_UNDEFINED_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 51: `SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_REQ_T` – Unregister Undefined Notify Request Packet

5.5.9 Unregister Undefined Notify Confirmation

The stack returns this packet to the application after it has processed the *Unregister Undefined Notify Request* packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_DATA_T;

typedef struct SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_DATA_T  tData;
} SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A4B	SIII_SL_IDN_CMD_UNREGISTER_UNDEFINED_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 52: `SIII_SL_IDN_UNREGISTER_UNDEFINED_NOTIFY_CNF_T` – Unregister Undefined Notify Confirmation Packet

5.5.10 Unregister IDN Notify Request

The packet SIII_SL_IDN_CMD_UNREGISTER_IDN_NOTIFY_REQ/CNF is used to unregister a notification occurring on read or write access of an IDN which has previously been established using the SIII_SL_IDN_CMD_REGISTER_IDN_NOTIFY_REQ packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_DATA_T;

typedef struct SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_DATA_T  tData;
} SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_T;
```

Packet Description

Structure SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_IDN	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A46	SIII_SL_IDN_CMD_UNREGISTER_IDN_NOTIFY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_DATA_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 53: SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_REQ_T – Unregister IDN Notify Request Packet

5.5.11 Unregister IDN Notify Confirmation

This packet is returned from the stack to the application after the stack has processed the Unregister IDN Notify Request packet.

Packet Structure Reference

```
typedef struct SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_DATA_Ttag
{
  TLR_UINT32                                    ulIdn;
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_DATA_T;

typedef struct SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_DATA_T  tData;
} SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_T;
```

Packet Description

Structure SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	5	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x5A47	SIII_SL_IDN_CMD_UNREGISTER_IDN_NOTIFY_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_T			
ulIdn	UINT32	Valid IDN	IDN to be accessed. The value must be a valid IDN according to the sercos specification.
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 54: SIII_SL_IDN_UNREGISTER_IDN_NOTIFY_CNF_T – Unregister IDN Notify Confirmation Packet

5.6 Diagnosis Handling

If a diagnosis has been reported, there are several places where the diagnosis can be found. Each reported diagnosis will be added to the trace buffer S-0-1303.0.10. The highest diagnosis message is also added to S-0-0390 (Diagnostic number) IDN. If the diagnosis contains the class “error” or “warning” some additional flags must be served within the RTData. The IDNs S-0-1045 and S-0-1500.x.02 contain the C1D and C2D flags and both IDNs are also mapped into the RTData. S-0-1045 will be completely handled by the slave stack (IDN and RTData handling). But in most cases, S-0-1502.x.02 must be served by application (dependent on set-configuration `ulFlags`).

5.6.1 Diagnosis occurred in Sercos slave stack

If a diagnosis occurs in the slave stack, the slave will handle the most IDNs. Just the C1D and C2D flags within S-0-1500.x.02 must be handled by application. If one of these flags shall be set, the stack will send an indication. The indication will also be sent if the flag disappears.

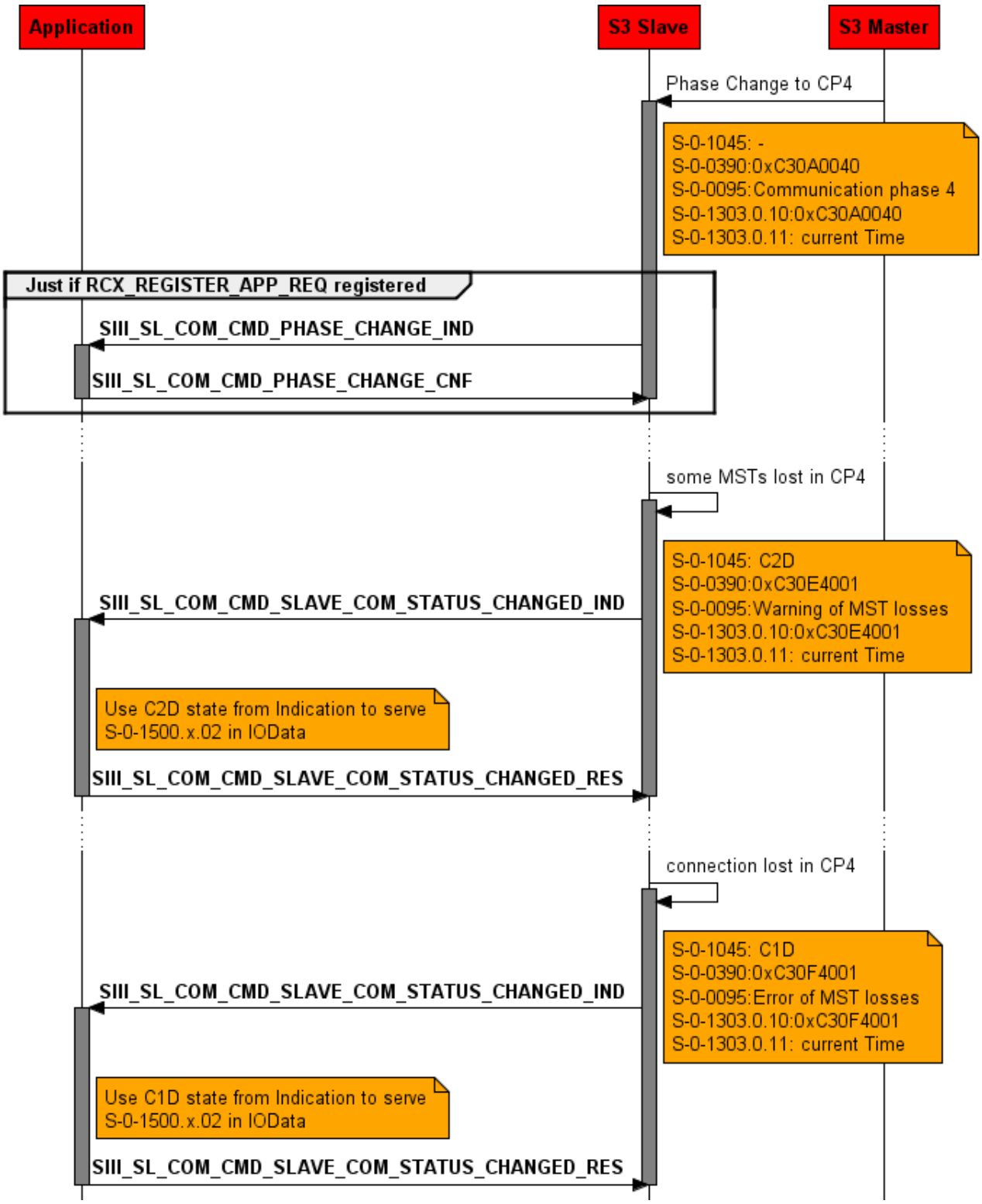


Figure 47: Diagnosis occurred in slave stack

5.6.2 Write Diagnostic Request

The Sercos slave administers diagnostic messages. If a new diagnosis occurs in the application the stack can get informed. If the stack receives a diagnosis from application the diagnosis will get promoted to the master. This is done by some flags and several IDNs (S-0-0095, S-0-0390, S-0-1303.0.10 and S-0-1303.0.11). The priorisation and handling is completely done by the stack. It is just necessary to report and remove diagnosis messages.

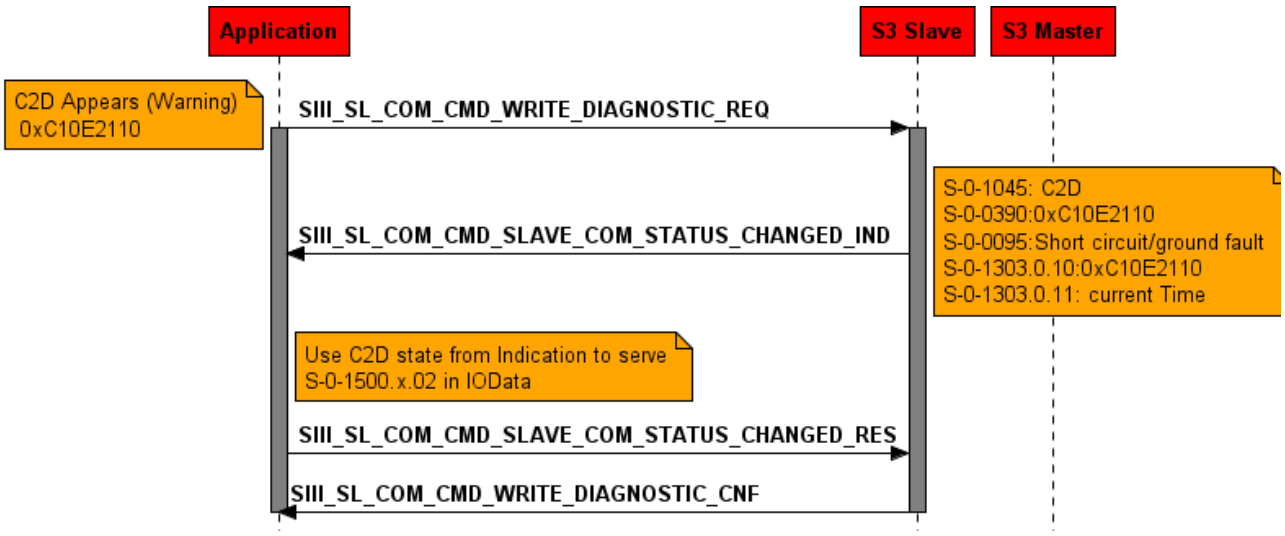


Figure 48: Write diagnostic request

`bClass` should have the same class as provided by the `ulDiagnosticNumber`

`abDiagnosticMessages` will not get evaluated for standard diagnosis codes. The stack has an internal database for standard diagnosis codes (>V3.1.30.0).

Packet Structure Reference

```
typedef struct SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_DATA_Ttag
{
  TLR_UINT8                                 bSlaveIdx;
  /* only used if Diagnostic number is fully manufacturer specific */
  TLR_UINT8                                 bClass;
  /* NULL-terminated string unless all characters are used */
  TLR_UINT8                                 abDiagnosticMessage[100];
  /* more data depending on S-0-1303.0.1 */
  TLR_UINT32                                ulDiagnosticNumber;
} SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_DATA_T;

#define SIII_SL_COM_DIAGNOSTIC_CLASS_INFO                                0x09
#define SIII_SL_COM_DIAGNOSTIC_CLASS_OPERATIONAL_STATE                   0x0A
#define SIII_SL_COM_DIAGNOSTIC_CLASS_RESERVED                            0x0B
#define SIII_SL_COM_DIAGNOSTIC_CLASS_PROCEDURE_COMMAND_SPECIFIC_STATE    0x0C
#define SIII_SL_COM_DIAGNOSTIC_CLASS_RESERVED2                           0x0D
#define SIII_CL_COM_DIAGNOSTIC_CLASS_WARNINGS                            0x0E
#define SIII_SL_COM_DIAGNOSTIC_CLASS_FAULTS                              0x0F

typedef struct SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_DATA_T   tData;
} SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_T;
```

Packet Description

Structure SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_COM	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	106	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3290	SIII_SL_COM_CMD_WRITE_DIAGNOSTIC_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
bClass	UINT8	0..15	Class
abDiagnosticMessage[100]	UINT8[]		Diagnostic message
ulDiagnosticNumber	UINT32		Diagnostic number

Table 55: `SIII_SL_COM_WRITE_DIAGNOSTIC_REQ_T` – Write Diagnostic Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_DATA_Ttag
{
  TLR_UINT8                                bSlaveIdx;
} SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_DATA_T;

typedef struct SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_Ttag
{
  TLR_PACKET_HEADER_T                      tHead;
  SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_DATA_T  tData;
} SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_T;
```

Packet Description

Structure SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3291	SIII_SL_COM_CMD_WRITE_DIAGNOSTIC_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 56: SIII_SL_COM_WRITE_DIAGNOSTIC_CNF_T – Write Diagnostic Confirmation Packet

5.6.3 Remove Diagnostic Request

Note: It is not possible to remove a stack internal diagnosis or a diagnosis reported by another application by this packet.

This packet is used to remove diagnosis messages previously reported by the application via SIII_SL_COM_WRITE_DIAGNOSTIC_REQ.

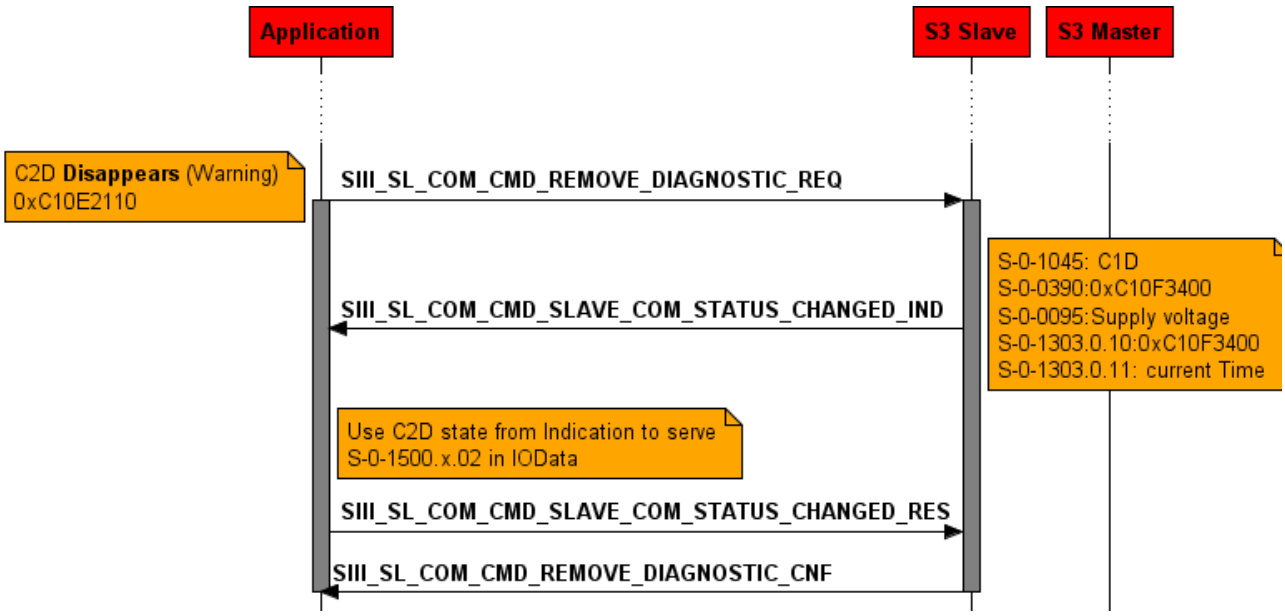


Figure 49: Remove diagnosis event warning

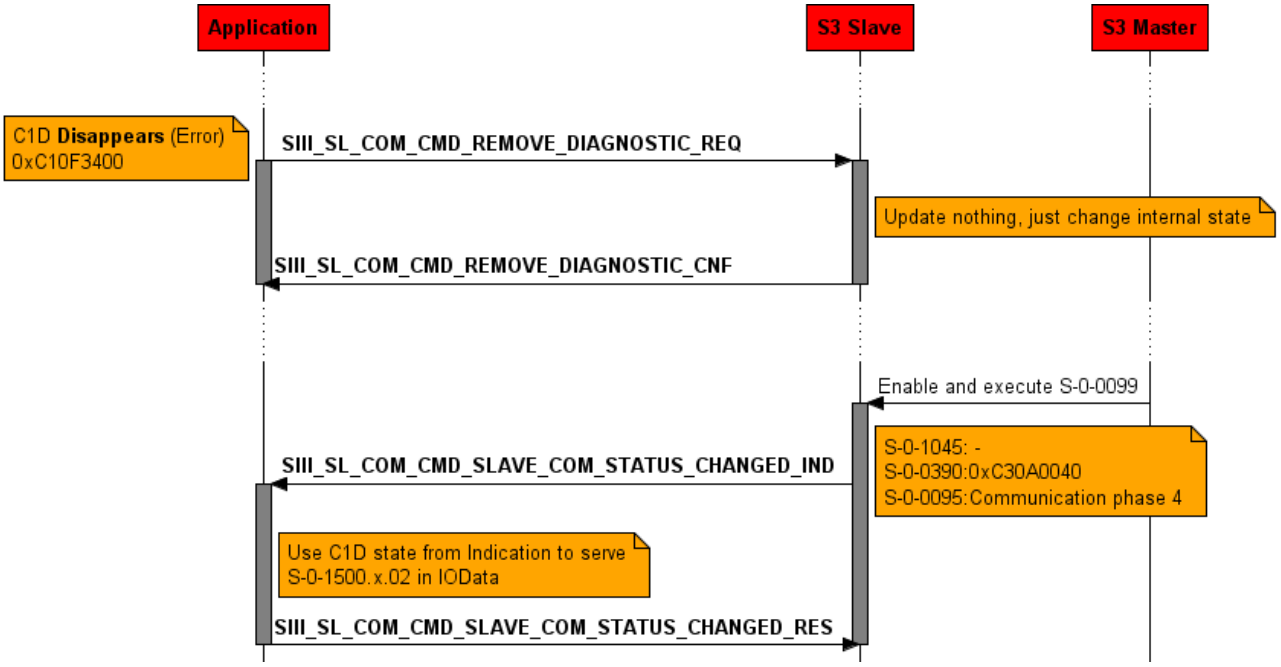


Figure 50: Remove diagnostic event error

A C2D diagnosis will be removed directly. But a C1D diagnosis will be removed if the master performs S-0-0099. If the application removes a C2D diagnosis, also the stack could have some C2D diagnosis messages active, so not in all cases the application will receive a “COM status change indication packet” (see section *COM Status Changed Indication* on page 172). Also if the master performs IDN S-0-0099, if the stack has some C1D diagnosis messages active, no “COM status change indication packet” is sent to the application.

The following table shows the available values for `ulFunctionFlag`:

Value	Symbolic Name	Meaning
0x00000000	SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_SPECIFIC_DIAGNOSIS	Erase one diagnosis message (the oldest if more are available) where the ulDiagnosticNumber matches (reported by this application previously)
0x00000001	SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_ERRORS	Erase all errors (reported by this application previously)
0x00000002	SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_WARNINGS	Erase all warnings (reported by this application previously)
0x00000004	SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_PROCEDURE_COMMAND_SPECIFIC_STATES	Erase all procedure command specific states (reported by this application previously)
0x00000008	SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_OPERATIONAL_STATES	Erase all operational states (reported by this application previously)
0x00000010	SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_SPECIFIC_DIAGNOSIS	Erase all diagnosis messages where the ulDiagnostic Number matches (reported by this application previously)

Table 57: Available values for `ulFunctionFlag`

Multiple flags can be set by simply adding them.

We recommend to remove each disappearing error/warning with the `ulFunctionFlag` `SIII_SL_COM_DIAGNOSTIC_FUNCTION_ERASE_SPECIFIC_DIAGNOSIS` and its diagnostic number equal given at the `SIII_SL_COM_WRITE_DIAGNOSTIC_REQ` packet. But it is also possible to remove all diagnosis (warnings and/or errors) reported by this application.

Packet Structure Reference

```
typedef struct SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_DATA_Ttag
{
  TLR_UINT8   bSlaveIdx;          /* the slave a diagnosis shall be removed */
  TLR_UINT32  ulFunctionFlags;    /* see defines, action that shall be done */
  TLR_UINT32  ulDiagnosticNumber; /* error code that shall be removed */
  /* more data depending on S-0-1303.0.1 */
} SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_DATA_T;

#define SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_SPECIFIC_DIAGNOSIS
0x00000000 /* Erase all diagnosis where the ulDiagnosticNumber matches (reported by this
application previously) */
#define SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_ERRORS
0x00000001 /* Erase all errors (reported by this application previously) */
#define SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_WARNINGS
0x00000002 /* Erase all warnings (reported by this application previously) */
#define SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_PROCEDURE_COMMAND_SPECIFIC_STATES
0x00000004 /* Erase all procedure command specific states (reported by this application
previously) */
#define SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL_OPERATIONAL_STATES
0x00000008 /* Erase all operational states (reported by this application previously) */
#define MSK_SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_ALL
0x0000000F

typedef struct SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_Ttag
{
```

```
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_DATA_T           tData;
} SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_T;
```

Packet Description

Structure SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_COM	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	106	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32AC	SIII_SL_COM_CMD_REMOVE_DIAGNOSTIC_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
ulFunctionFlags	UINT	0..15	See *Table 57: Available values for ulFunctionFlag* and description above
ulDiagnosticNumber	UINT32		Diagnostic number which shall be removed. Just evaluated if the ulFunctionFlags is set to SIII_SL_COM_DIAGNOSTIC_FUNCTION_REMOVE_SPECIFIC_DIAGNOSIS

Table 58: SIII_SL_COM_REMOVE_DIAGNOSTIC_REQ_T – Remove Diagnostic Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_DATA_Ttag
{
  TLR_UINT8                                bSlaveIdx;
} SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_DATA_T;

typedef struct SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_Ttag
{
  TLR_PACKET_HEADER_T                      tHead;
  SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_DATA_T  tData;
} SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_T;
```

Packet Description

Structure SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32AD	SIII_SL_COM_CMD_REMOVE_DIAGNOSTIC_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 59: SIII_SL_COM_REMOVE_DIAGNOSTIC_CNF_T –Remove Diagnostic Confirmation Packet

5.6.4 Summary of the stack diagnosis handling

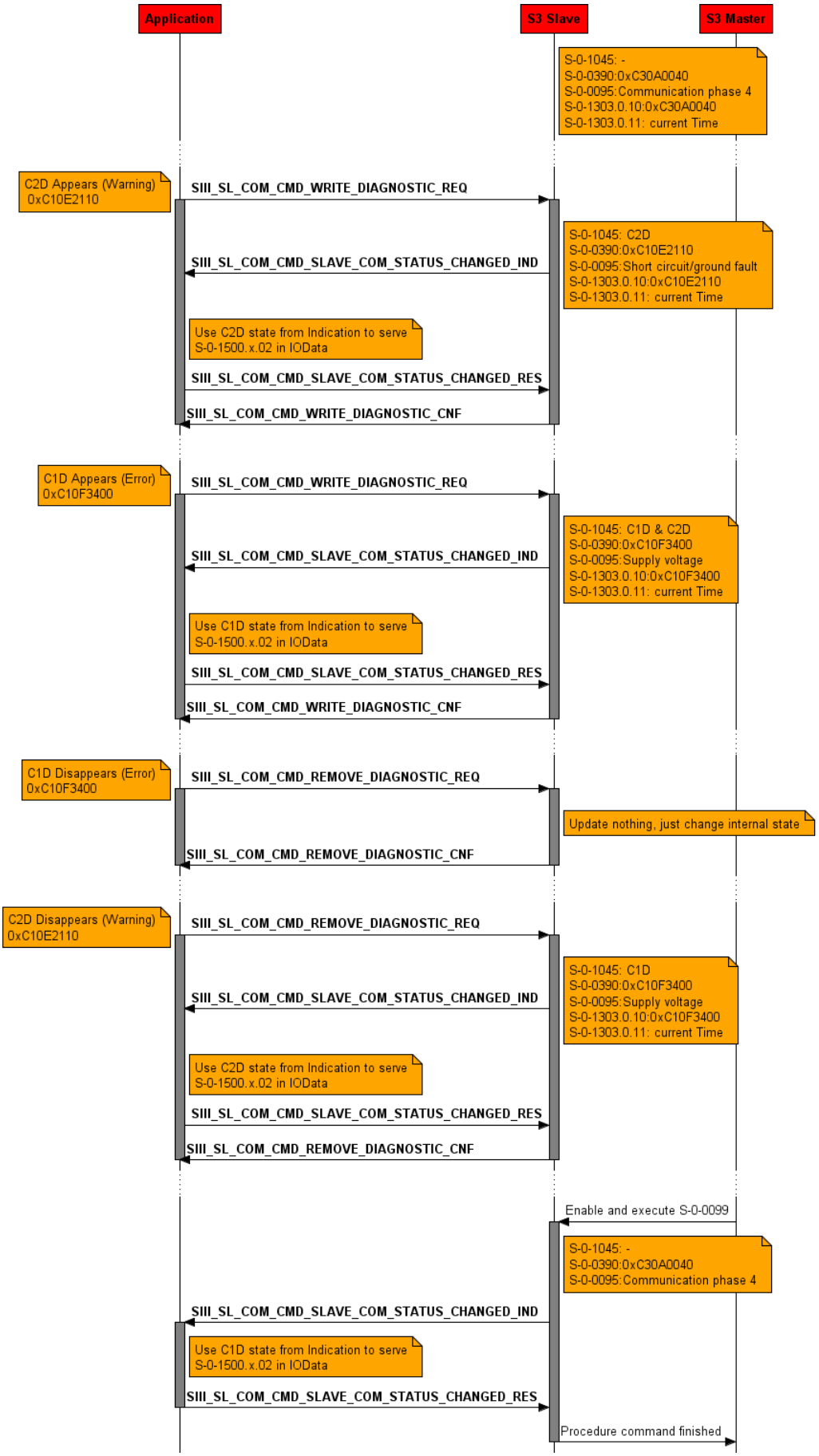


Figure 51: Overview of diagnosis handling

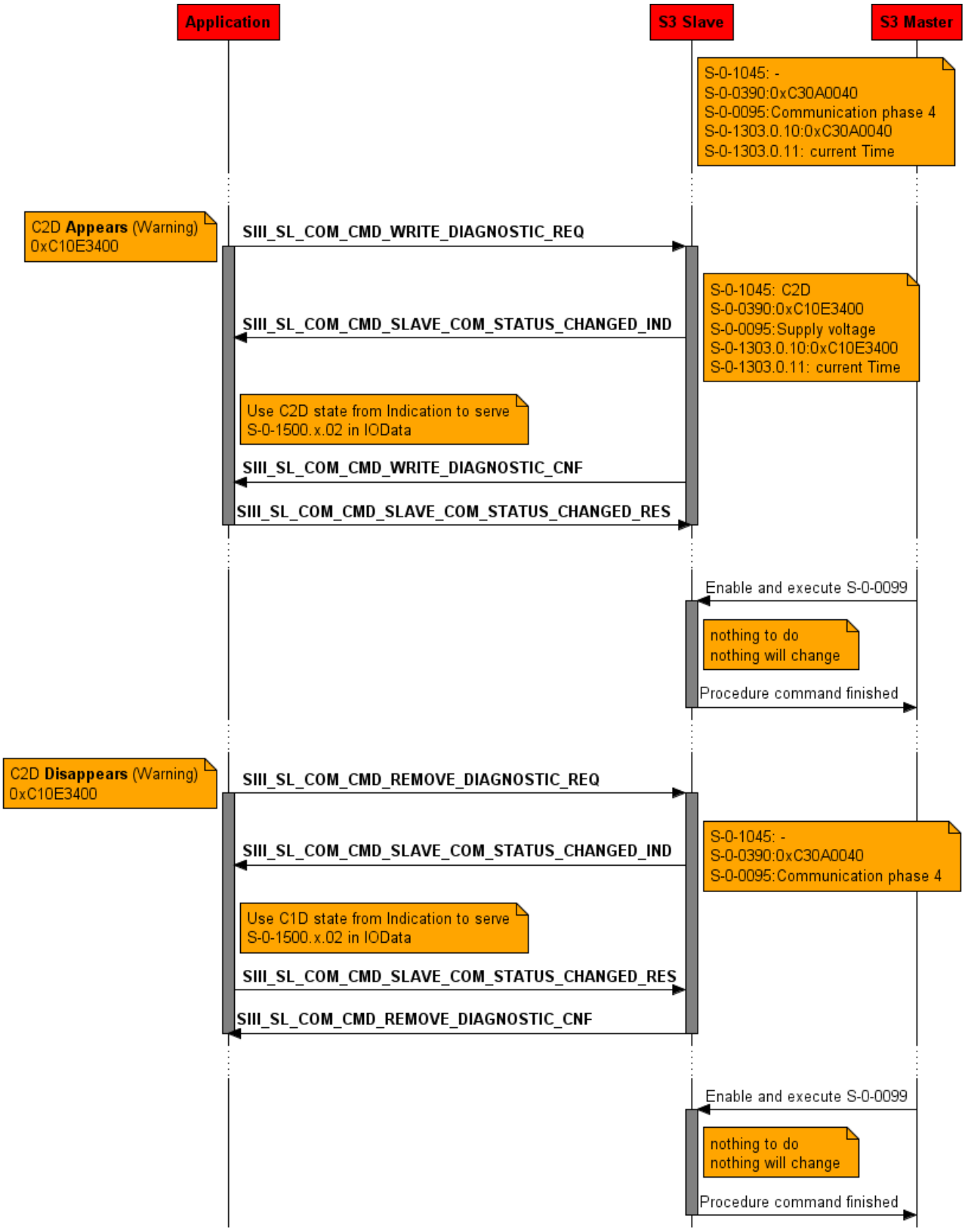


Figure 52: C2D (Warning) Example

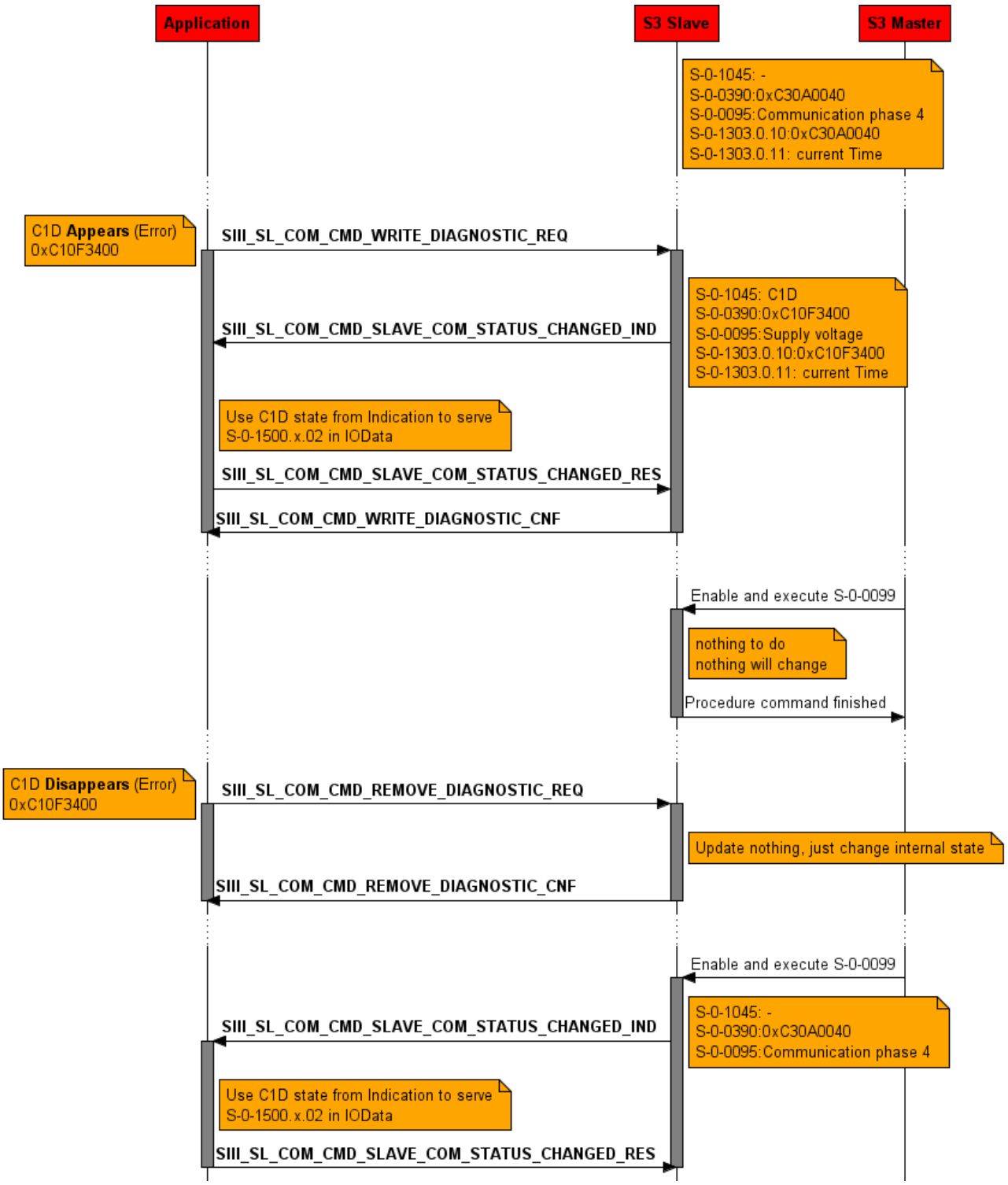


Figure 53: C1D (Error) Example

5.7 Procedure Commands

Procedure commands are special Sercos IDNs. These procedure command IDNs are used to start a mechanism on the slave which may run a long, not defined, time. Within the startup sequence there are two special procedure commands which could be relevant for your application (S-0-0127 and S-0-0128).

- Before phase transition from CP2 to CP3 the master performs the procedure command S-0-0127.
- Before phase transition from CP3 to CP4 the master performs the procedure command S-0-0128.

If one of these procedure commands is executed, the slave has to check all necessary parameters and data for the next phase. It shall be checked whether all data are available and whether they are in a valid range. The slave does all these checks for the IDNs created by default.

Just if your application has created own IDNs and these IDNs must be checked at a phase transition, the application should register for this Indications. If the registration is done three indications must be handled by application.

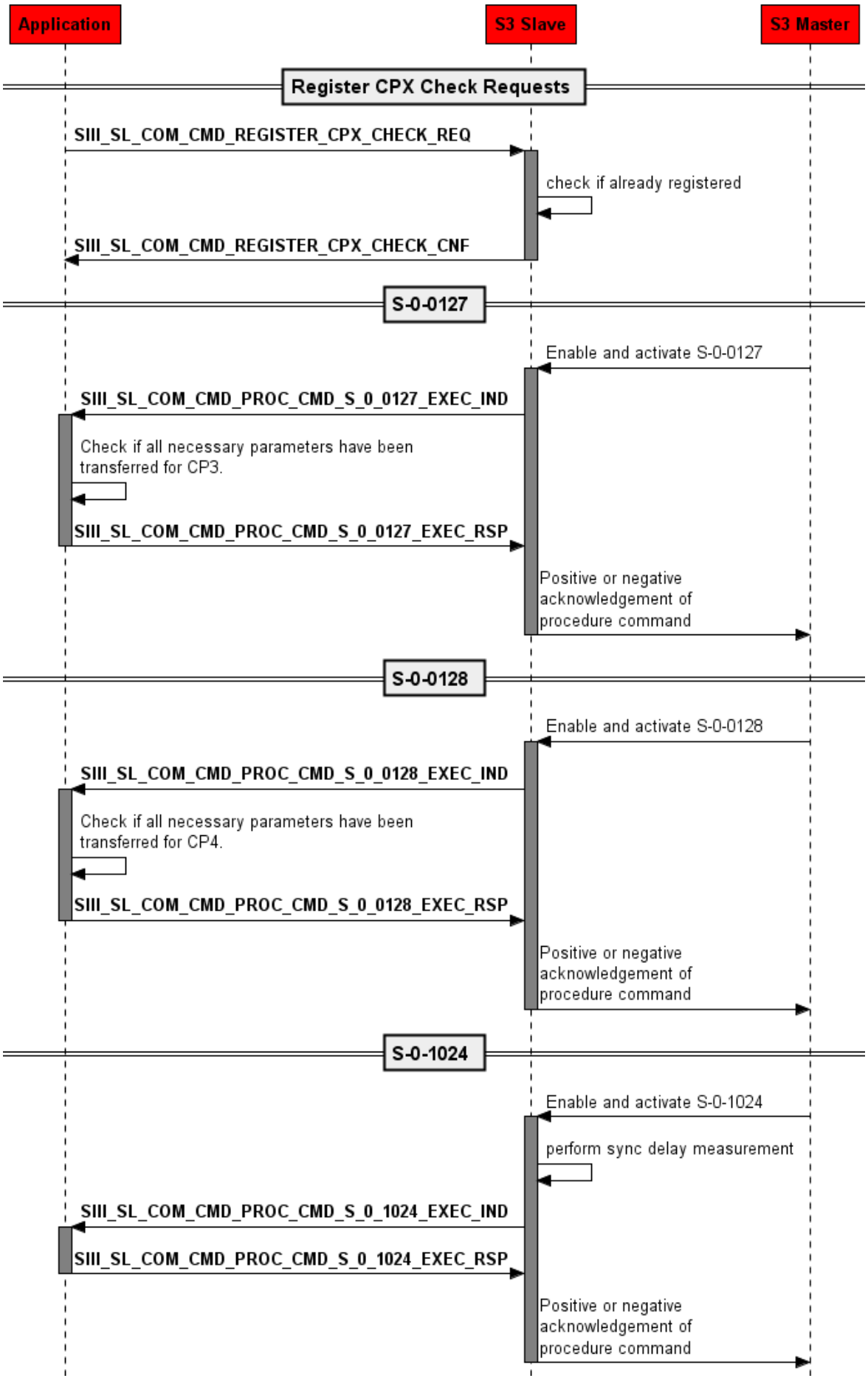


Figure 54: Register CPX check indications

5.7.1 Register CPX Check Indications

The packet SIII_SL_COM_CMD_REGISTER_CPX_CHECK_REQ/CNF allows you to register for reception of the following indications related to procedure commands:

- Execute Procedure Command S-0-0127 Indication (see page 138)
- Execute Procedure Command S-0-0128 Indication (see page 145)
- Execute Procedure Command S-0-1024 Indication (see page 148)

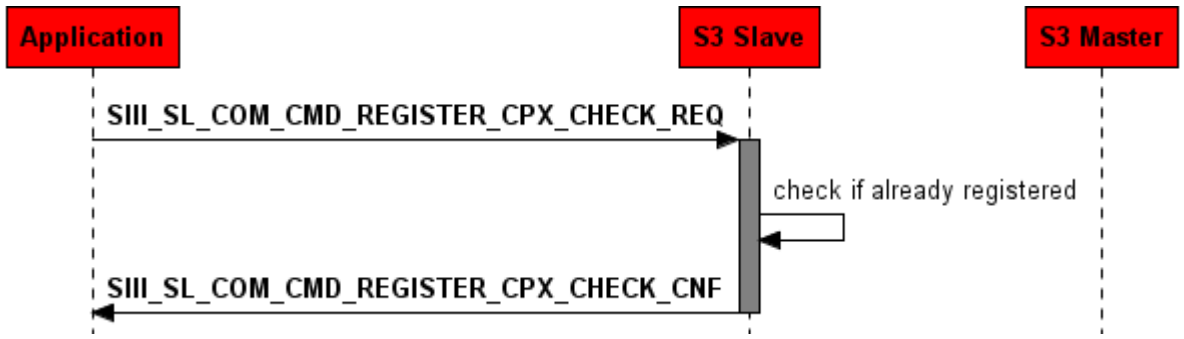


Figure 55: Register CPX Check Indications

Packet Structure Reference

```
typedef struct SIII_SL_COM_REGISTER_CPX_CHECK_REQ_DATA_Ttag
{
  TLR_UINT8                                 bSlaveIdx;
} SIII_SL_COM_REGISTER_CPX_CHECK_REQ_DATA_T;

typedef struct SIII_SL_COM_REGISTER_CPX_CHECK_REQ_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  SIII_SL_COM_REGISTER_CPX_CHECK_REQ_DATA_T  tData;
} SIII_SL_COM_REGISTER_CPX_CHECK_REQ_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_CPX_CHECK_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_COM	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3240	SIII_SL_COM_CMD_REGISTER_CPX_CHECK_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_CPX_CHECK_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 60: `SIII_SL_COM_REGISTER_CPX_CHECK_REQ_T` – Register CPX Check Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_REGISTER_CPX_CHECK_CNF_DATA_Ttag
{
  TLR_UINT8                                   bSlaveIdx;
} SIII_SL_COM_REGISTER_CPX_CHECK_CNF_DATA_T;

typedef struct SIII_SL_COM_REGISTER_CPX_CHECK_CNF_Ttag
{
  TLR_PACKET_HEADER_T                         tHead;
  SIII_SL_COM_REGISTER_CPX_CHECK_CNF_DATA_T   tData;
} SIII_SL_COM_REGISTER_CPX_CHECK_CNF_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_CPX_CHECK_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3241	SIII_SL_COM_CMD_REGISTER_CPX_CHECK_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_CPX_CHECK_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 61: SIII_SL_COM_REGISTER_CPX_CHECK_CNF_T – Register CPX Check Confirmation Packet

5.7.2 Execute Procedure Command S-0-0127 Indication

The indication `SIII_SL_COM_CMD_PROC_CMD_S_0_0127_EXEC_IND` occurs every time a S-0-0127 procedure command has to be executed. This is the case if the master has successfully performed all preparations for the transition from CP2 to CP3 (page 26) and requests the slave to check for completeness of all transferred parameters.

This packet is relevant only if you have to check some parameters.

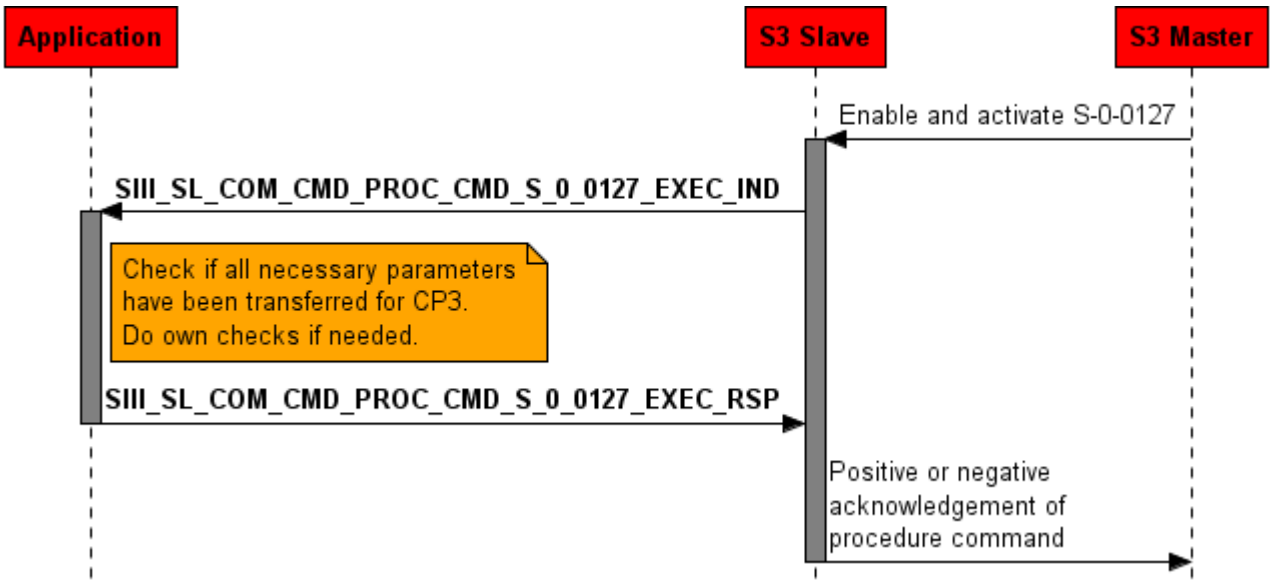


Figure 56: S-0-0127 Execute Indication

The user has the following possibilities:

- Reconfigure the layout of the data in the DPM. The initial configuration was done with the command `SIII_SL_AP_CMD_SET_CONFIGURATION_REQ`. Reconfiguration may become necessary if several slaves are used and there are master configured connections which collide with the DPM layout or the master configured Cross Communication. Usually it is not needed.
- Reconfigure the ConClk and reconfigure the DivClk. Background: now the used bus cycle time for CP3 and CP4 is known.
- Write a list of application based IDNs which are configured incorrectly by the master into `aulS_0_0021[]`. This list is merged with incorrectly configured communication based IDNs and stored in S-0-21.

The indication packet has the following parameters:

- The slave index is delivered within the variable `bSlaveIdx`.
- The communication cycle time (t_{Scyc}) is stored in variable `ulCommunicationCycle` and describes the basic cycle time for communication in nanoseconds. It is relevant for CP3 and CP4. It relates to IDN S-0-1002. It defines the time intervals for the transmission of cyclic data. The allowed cycle times are defined as 31,25 µs, 62,5 µs, 125 µs, 250 µs, 500 µs, 750 µs, 1 ms, 1,25 ms ... up to 65 ms in steps of 250 µs. Typically, in CP2, the master needs to transfer t_{Scyc} to the slave. Then t_{Scyc} has to be activated in CP3 and CP4. Values of the communication cycle time (t_{Scyc}) **below 250 µs** are currently **not supported**.
- The AT Transmission Starting Time is stored in variable `ulAtTransmissionStartTime` and is defined as the time when the Sercos Master starts sending the AT0 telegram. In the

Sercos specification this value is often denominated as t_1.This item corresponds to IDN S-0-1006. It is represented by an unsigned decimal value with 3 places after the decimal point. It is specified in units of microseconds. The minimum value is 0, the maximum value t_{Scyc}.

- The connection info `SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_IND_T` is present for each connection of the slave. For a more detailed description of the connection parameters, have a look at *Table 24* on page 68. It is structured like:

Parameter	Type	Meaning	Values
usConnectionSetup	UINT16	Connection setup	
usProcImgConnCtrlOffset	UINT16	Connection Control offset specifying the position of the offset of the connection control process image within the DPM.	0…5758
usProcImgRtDataOffset	UINT16	Real-time data process image offset	See notes 1, 2 on page 68.
usProcImgMaxRtDataLength	UINT16	Real time data maximum length	See notes 1, 2 and 3 on page 68.

Table 62: `SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_IND_T` – parameters, their meanings and their ranges of allowed values

Packet Structure Reference

```
#define SIII_SL_COM_PROC_CMD_S_0_0127_NUM_CONNECTIONS       4

typedef struct SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_IND_Ttag
{
  TLR_UINT16                                    usConnectionSetup;
  TLR_UINT16                                    usProcImgConnCtrlOffset;
  TLR_UINT16                                    usProcImgRtDataOffset;
  TLR_UINT16                                    usProcImgMaxRtDataLength;
} SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_IND_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
  TLR_UINT32                                    ulCommunicationCycle;
  TLR_UINT32                                    ulAtTransmissionStartTime;
  /* in ns, only valid if SCP_Sync is enabled */
  SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_IND_T
atConnInfo[SIII_SL_COM_PROC_CMD_S_0_0127_NUM_CONNECTIONS];
} SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	41	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3250	SIII_SL_COM_CMD_PROC_CMD_S_0_0127_EXEC_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
ulCommunicationCycle	UINT32	31250.. 65000000	S-0-1002 Communication cycle time (CP3/4) (this is the value denominated as t_{Scyc} in the Sercos specifications). This value is specified in nanoseconds. Minimum Value 31 250 (31,250 µs), Maximum Value 65 000 000 (65000,000 µs)
ulAtTransmissionStartTime	UINT32	0.. ulCommunicationCycle	S-0-1006 AT transmission startup time This value is specified in nanoseconds.
atConnInfo[4]	SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_IND_T[] Connection info data structure.		

Table 63: `SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_IND_T` – Execute Procedure Command `S_0_0127` Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_RES_Ttag
{
  TLR_UINT16                                    usProcImgConnCtrlOffset;
  TLR_UINT16                                    usProcImgRtDataOffset;
  TLR_UINT16                                    usProcImgMaxRtDataLength;
} SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_RES_T;

#define SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_MAX_IDNS_PER_SEGMENT (1024 /
sizeof(UINT32))
#define SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_MIN_SIZE
(sizeof(SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_T) - sizeof(TLR_UINT32) *
SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_MAX_IDNS_PER_SEGMENT)

typedef struct SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
  TLR_BOOLEAN32                                 fModifyConnInfo;
  TLR_BOOLEAN32                                 fModifySyncConfiguration;

  /* sync configuration:
       last received configuration is valid as it is shared
       between all slaves in a device.
   */
  TLR_UINT32                                    ulDivModeControl;
  TLR_UINT32                                    ulConClkLength;
  TLR_UINT32                                    ulTDivClk;
  TLR_UINT32                                    ulDTDivClk;
  TLR_UINT32                                    ulDivClkLength;
  TLR_UINT32                                    ulNDivClk;
  TLR_UINT32                                    ulTInt2;
  TLR_UINT32                                    ulTInt3;

  /* connection info - configuration per slave  */
  SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_RES_T
atConnInfo[SIII_SL_COM_PROC_CMD_S_0_0127_NUM_CONNECTIONS];

  TLR_UINT32
aulS_0_0021_List[SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_MAX_IDNS_PER_SEGMENT];
} SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_T;

#define SIII_SL_COM_PROC_CMD_S_0_0127_DIV_MODE_CONTROL_DISABLED               0
#define SIII_SL_COM_PROC_CMD_S_0_0127_DIV_MODE_CONTROL_SET_MODE_0             1
#define SIII_SL_COM_PROC_CMD_S_0_0127_DIV_MODE_CONTROL_SET_MODE_1             2

typedef struct SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_T;
```

The connection info `SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_RES_T` is present for each connection of the slave. For a more detailed description of the connection parameters, have a look at Table 24 on page 68.

It is structured like:

Parameter	Type	Meaning	Values
usProcImgConnCtrlOffset	UINT16	Offset of the connection control process image within the DPM.	0…5758
usProcImgRtDataOffset	UINT16	Real-time data process image offset	See notes 1,2 on page 68.
usProcImgMaxRtDataLength	UINT16	Real time data maximum length	See notes 1, 2 and 3 on page 68.

Table 64: `SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_RES_T` – parameters, their meanings and their ranges of allowed values

The `aulS_0_0021_List[256]` is used to store the list of all parameters which caused an error during switching from CP2 to CP3. The content of this list is stored in IDN S-0-0021. If no error occurred, the list is empty.

The possible values of `ulDivModeControl` in the response packet are defined in the following table:

Value	Description
0	SIII_SL_COM_PROC_CMD_S_0_0127_DIV_MODE_CONTROL_DISABLED DivMode control disabled
1	SIII_SL_COM_PROC_CMD_S_0_0127_DIV_MODE_CONTROL_SET_MODE_0 DivMode control mode 0
2	SIII_SL_COM_PROC_CMD_S_0_0127_DIV_MODE_CONTROL_SET_MODE_1 DivMode control mode 1

Table 65: Possible Values of `ulDivModeControl` in the Response Packet

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	65..1089	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3251	SIII_SL_COM_CMD_PROC_CMD_S_0_0127_EXEC_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
fModifyConnInfo	BOOLEAN32		Connection info modified
fModifySyncConfiguration	BOOLEAN32		Sync configuration modified
ulDivModeControl	UINT32	0..2	DivMode Control
ulConClkLength	UINT32		ConClk length
ulTDivClk	UINT32		Contents of TDivClk register
ulDTDivClk	UINT32		since V3.1.X: not used
ulDivClkLength	UINT32		since V3.1.X: not used
ulNDivClk	UINT32		Contents of NDivClk register
ulTInt2	UINT32		TInt2 (not used)
ulTInt3	UINT32		Tint3 (not used)
atConnInfo[4]	SIII_SL_COM_PROC_CMD_S_0_0127_CONN_INFO_RES_T[] Connection info		
aulS_0_0021_List[256]	UINT32[]		IDN-list of invalid operation data for CP2 (IDN S-0-0021)

Table 66: `SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_T` – Execute Procedure Command `S_0_0127` Response Packet

5.7.3 Execute Procedure Command S-0-0128 Indication

The indication `SIII_SL_COM_CMD_PROC_CMD_S_0_0128_EXEC_IND` occurs every time an S-0-0128 procedure command has to be executed. This is the case if the master has successfully performed all preparations for the transition from CP3 to CP4 (see page 27) and requests the slave to check for completeness of all transferred parameters.

In normal cases nothing must be checked. Just send the packet back.

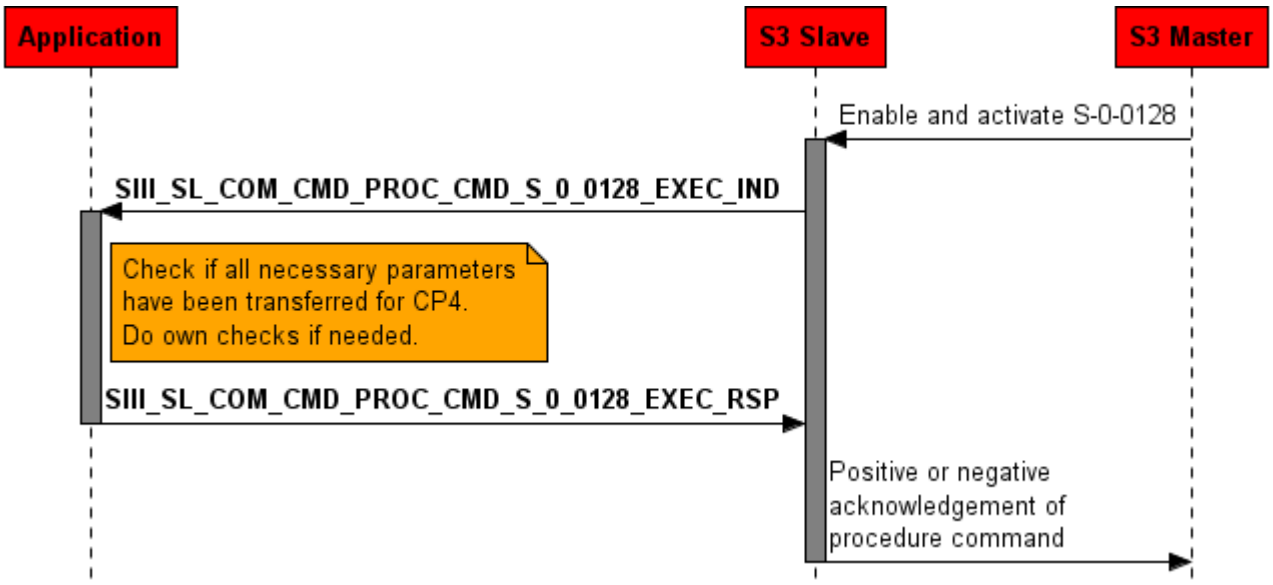


Figure 57: S-0-0128 Executed Indication

The `aulS_0_0022_List[256]` is used to store the list of all parameters which caused an error during the transition process switching from CP3 to CP4. The content of this list is stored in IDN S-0-0022. If no error occurred, the list is empty.

Packet Structure Reference

```
typedef struct SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3252	SIII_SL_COM_CMD_PROC_CMD_S_0_0128_EXEC_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 67: `SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_IND_T` – Execute Procedure Command `S_0_0128` Indication Packet

Packet Structure Reference

```
#define SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_DATA_MAX_IDNS_PER_SEGMENT    (1024 /
sizeof(UINT32))

typedef struct SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_DATA_Ttag
{
  TLR_UINT8                                        bSlaveIdx;
  TLR_UINT32
aulS_0_0022_List[SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_DATA_MAX_IDNS_PER_SEGMENT];
} SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_Ttag
{
  TLR_PACKET_HEADER_T                              tHead;
  SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1025	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3253	SIII_SL_COM_CMD_PROC_CMD_S_0_0128_EXEC_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_0127_EXEC_RES_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
aulS_0_0022_List[256]	UINT32[]		IDN-list of invalid operation data for CP3 (IDN S-0-0022)

Table 68: SIII_SL_COM_PROC_CMD_S_0_0128_EXEC_RES_T – Execute Procedure Command S_0_0128 Response Packet

5.7.4 Execute Procedure Command S-0-1024 Indication

Note: This packet is only required for slaves implementing SCP profile SCP_Sync. For a usual device nothing is to be done with this packet. Just return it to the master.

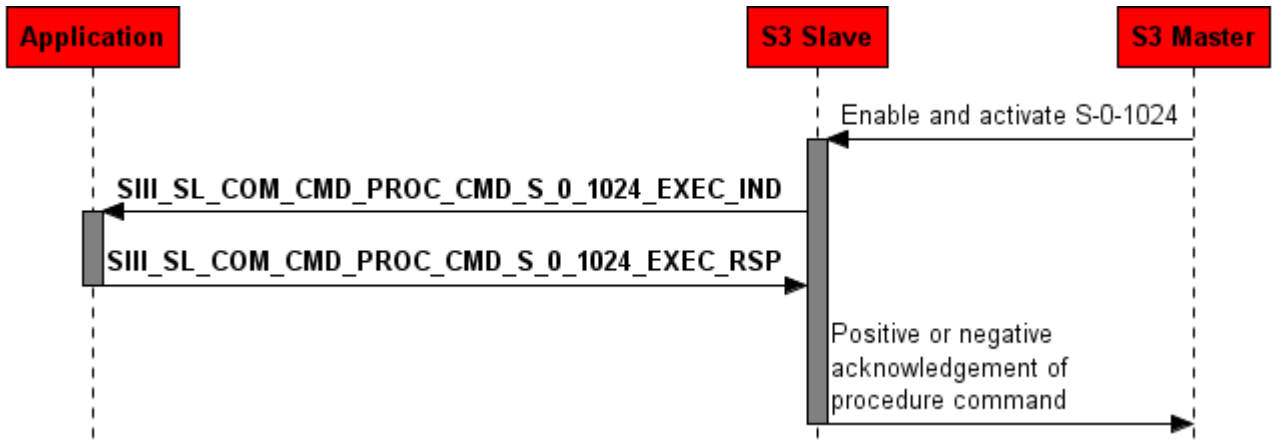


Figure 58: S-0-1024 Executed Indication

The indication occurs every time an IDN S-0-1024 procedure command has to be executed. This is the case if the Sercos Master requests the Sercos slave to perform a synchronization delay measuring procedure (in CP2, after ring healing and Hot-plug). Therefore this packet informs the application when the procedure is executed. No additional handling is required if this packet is received.

For more information

- on synchronization see section 5.5.3 *Synchronization* of the Sercos Communication specification V1.1.2.1.7.
- on the SCP profile SCP_Sync see section 4.3 SCP_Sync of the Sercos Communication specification V1.1.2.1.7.

Packet Structure Reference

```
typedef struct SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3254	SIII_SL_COM_CMD_PROC_CMD_S_0_1024_EXEC_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 69: `SIII_ SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_IND_T` – Execute Procedure Command S_0_1024 Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_DATA_T      tData;
} SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3255	SIII_SL_COM_CMD_PROC_CMD_S_0_1024_EXEC_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_1024_EXEC_RES_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 70: SL_COM_CMD_PROC_CMD_S_0_1024_EXEC_RES – Execute Procedure Command S-0-1024 Response Packet

5.7.5 Register for Procedure Command S-0-0099 Execution Indication

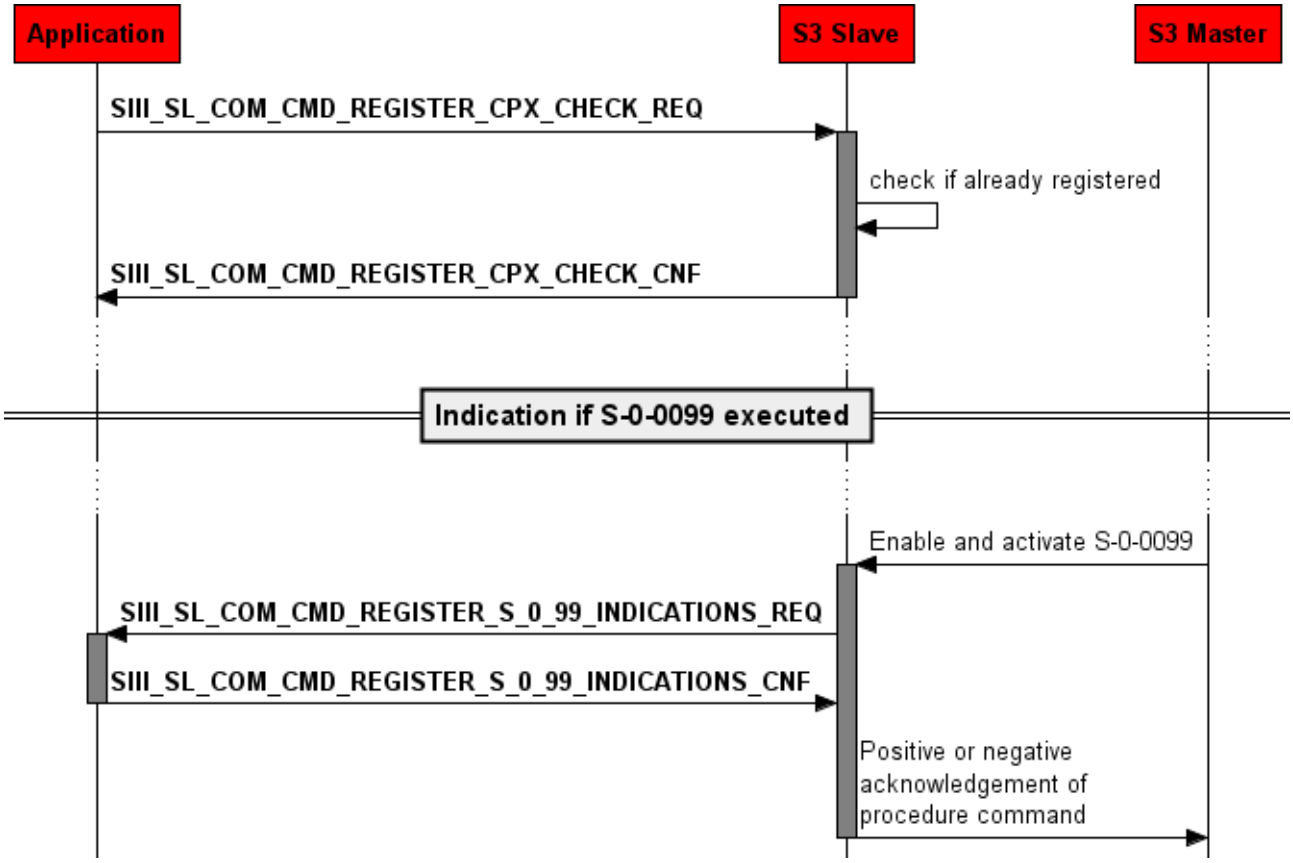


Figure 59: Register for Procedure Command S-0-0099 execution indication

With the packet `SIII_SL_COM_CMD_REGISTER_S_0_99_INDICATIONS_REQ` the user application can register for reception of S-0-0099 Reset class 1 diagnostic indications (`SIII_SL_COM_CMD_PROC_CMD_S_0_0099_EXEC_IND`).

Packet Structure Reference

```
typedef struct SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_DATA_T;

typedef struct SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_DATA_T  tData;
} SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_COM	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32A0	SIII_SL_COM_CMD_REGISTER_S_0_99_INDICATIONS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 71: `SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_REQ_T` – Register `S_0_99` Indications Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
} SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_DATA_T;

typedef struct SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_DATA_T  tData;
} SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32A1	SIII_SL_COM_CMD_REGISTER_S_0_99_INDICATIONS_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 72: `SIII_SL_COM_REGISTER_S_0_99_INDICATIONS_CNF_T` – Register `S_0_99` Indications Confirmation Packet

5.7.6 Execute Procedure Command S-0-0099 Indication

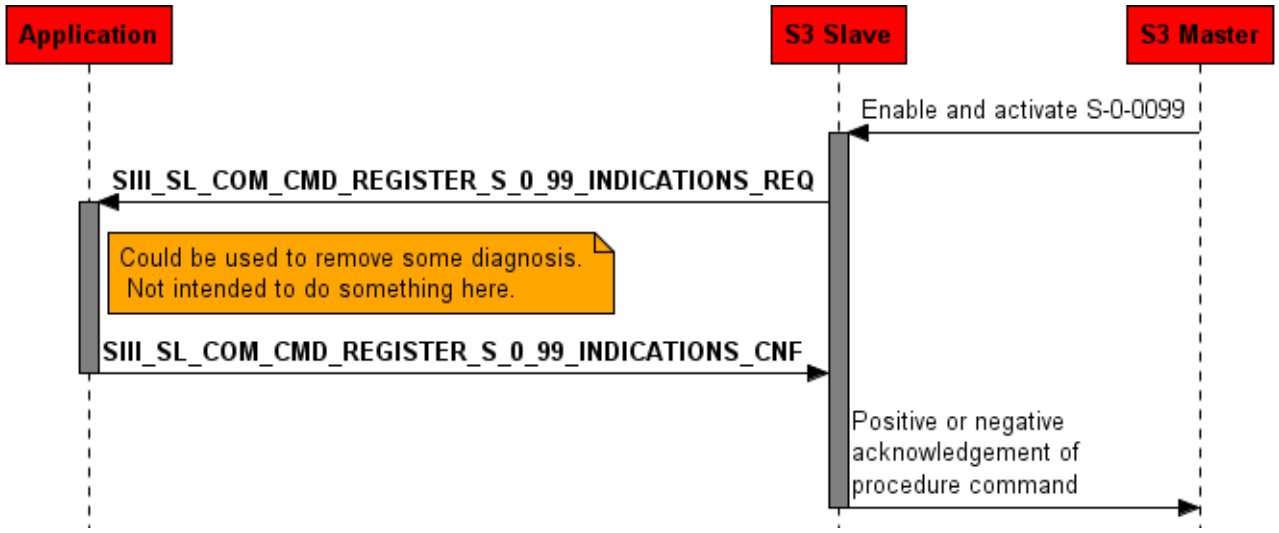


Figure 60: Procedure Command S-0-0099 execution indication

The packet `SIII_SL_COM_CMD_PROC_CMD_S_0_0099_EXEC_IND` is sent by the stack if the master executes the procedure command S-0-0099 Reset class 1 diagnostic (C1D). The indication must be answered with `SIII_SL_COM_CMD_PROC_CMD_S_0_0099_EXEC_RES` packet.

Packet Structure Reference

```
typedef struct SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32A4	SIII_SL_COM_CMD_PROC_CMD_S_0_0099_EXEC_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 73: SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_IND_T – Execute Procedure Command S_0_0099 Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
} SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_DATA_T;

typedef struct SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_DATA_T  tData;
} SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_T;
```

Packet Description

Structure SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x32A5	SIII_SL_COM_CMD_PROC_CMD_S_0_0099_EXEC_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 74: SIII_SL_COM_PROC_CMD_S_0_0099_EXEC_RES_T – Execute Procedure Command S_0_0099 Response Packet

5.8 Update Device Status

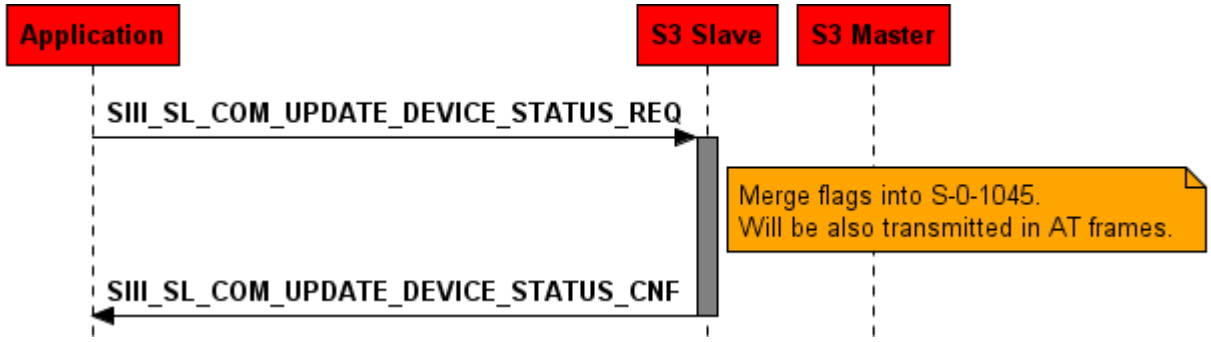


Figure 61: Update device status

Note: Do not use this packet to modify bit 6, 7 and 15! These bits are handled by the stack. They are just supported for compatibility reasons.

You may use this packet to:

- Modify bit 5 if you implemented procedure commands in your application
- Modify bit 9 if an error occurred during IOData exchange.

The following bits of the Device Status can be set using this packet:

Bit	Description
0-3	unused, set to zero
4	`MSK_SIII_SL_COM_DEVICE_STATUS_PARAMETERIZATION_LEVELS` Parameterization level
5	`MSK_SIII_SL_COM_DEVICE_STATUS_PROC_COMMAND_CHANGE_BIT` Procedure command change bit
6	`MSK_SIII_SL_COM_DEVICE_STATUS_C2D_WARNING_BIT` Warning of device (C2D)
7	`MSK_SIII_SL_COM_DEVICE_STATUS_C1D_ERROR_BIT` Error of device (C1D)
8	unused, set to zero
9	`MSK_SIII_SL_COM_DEVICE_STATUS_ERROR_OCCURED` Error connection
10-14	unused, set to zero
15	`MSK_SIII_SL_COM_DEVICE_STATUS_COMMUNICATION_WARNING` Communication warning interface

Table 75: Possible Values of `usDeviceStatus`

Packet Structure Reference

```
typedef struct SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_DATA_Ttag
{
  TLR_UINT8                                   bSlaveIdx;
  TLR_UINT16                                  usDeviceStatus;
  TLR_UINT16                                  usDeviceStatusModifyMask;
} SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_DATA_T;

typedef struct SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                         tHead;
  SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_DATA_T  tData;
} SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_T;
```

Packet Description

Structure SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_COM	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	5	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3264	SIII_SL_COM_CMD_UPDATE_DEVICE_STATUS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
usDeviceStatus	UINT16	Bit mask	Device Status (for meaning of bits see *Table 75: Possible Values of `usDeviceStatus`*)
usDeviceStatus ModifyMask	UINT16	Bit mask	Mask for the bits of the device status that shall be changed

Table 76: `SIII_SL_COM_UPDATE_DEVICE_STATUS_REQ_T` – Update Device Status Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_DATA_Ttag
{
  TLR_UINT8                                    bSlaveIdx;
} SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_DATA_T;

typedef struct SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_Ttag
{
  TLR_PACKET_HEADER_T                          tHead;
  SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_DATA_T  tData;
} SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_T;
```

Packet Description

Structure SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3265	SIII_SL_COM_CMD_UPDATE_DEVICE_STATUS_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 77: SIII_SL_COM_UPDATE_DEVICE_STATUS_CNF_T – Update Device Status Confirmation Packet

5.9 Changes of the IP Address

Just if the application wants to store the IP settings of the slave in a non volatile way this chapter must be read.

The Sercos Master configures IP settings through the IDNs

- S-0-1020 IP Address
- S-0-1021 Subnet Mask
- S-0-1022 Gateway Address
- S-0-1039 Hostname

and uses the procedure command S-0-1048 to activate the IP settings.

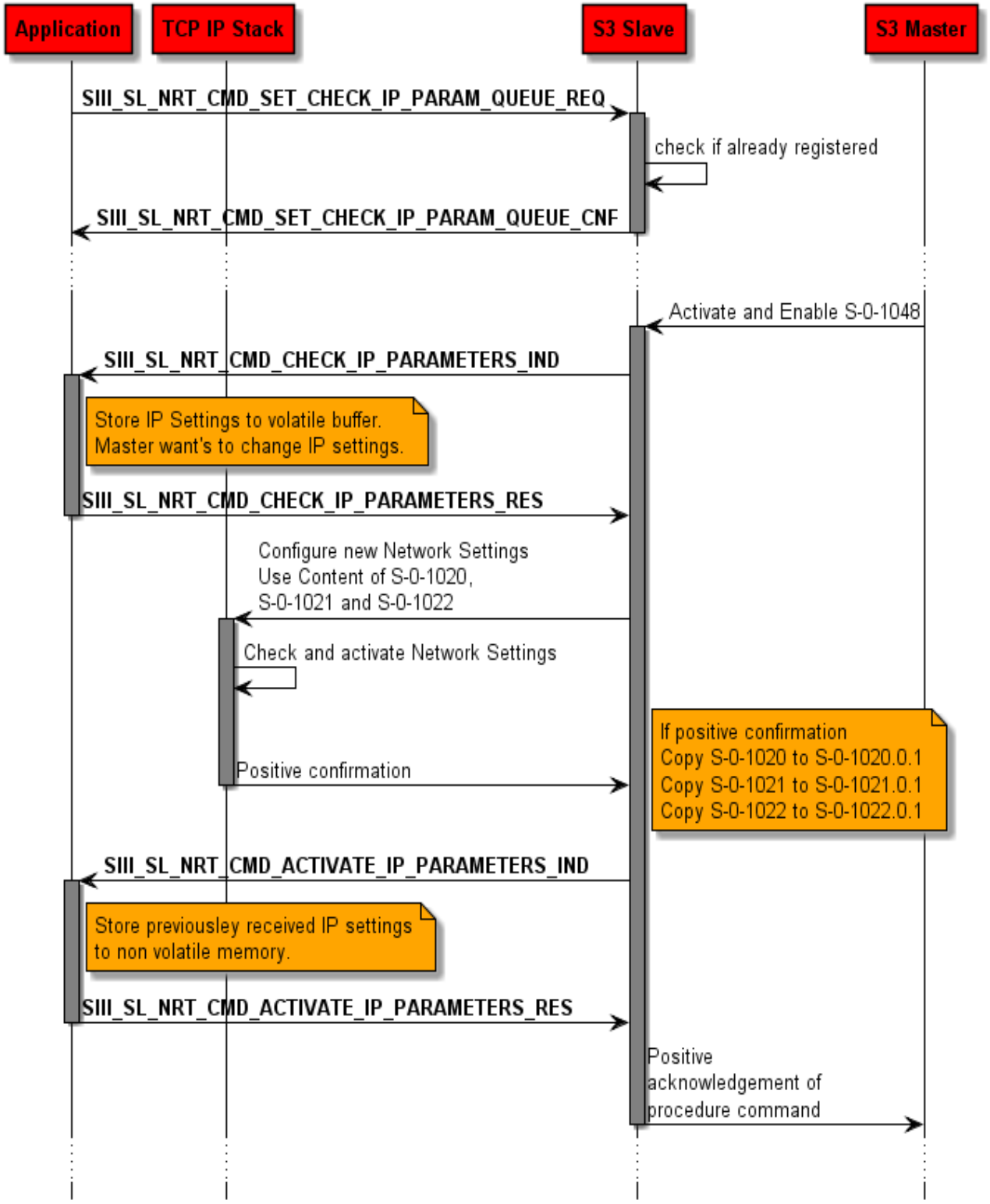


Figure 62: Changes of IP address complete overview

5.9.1 Register check IP parameter

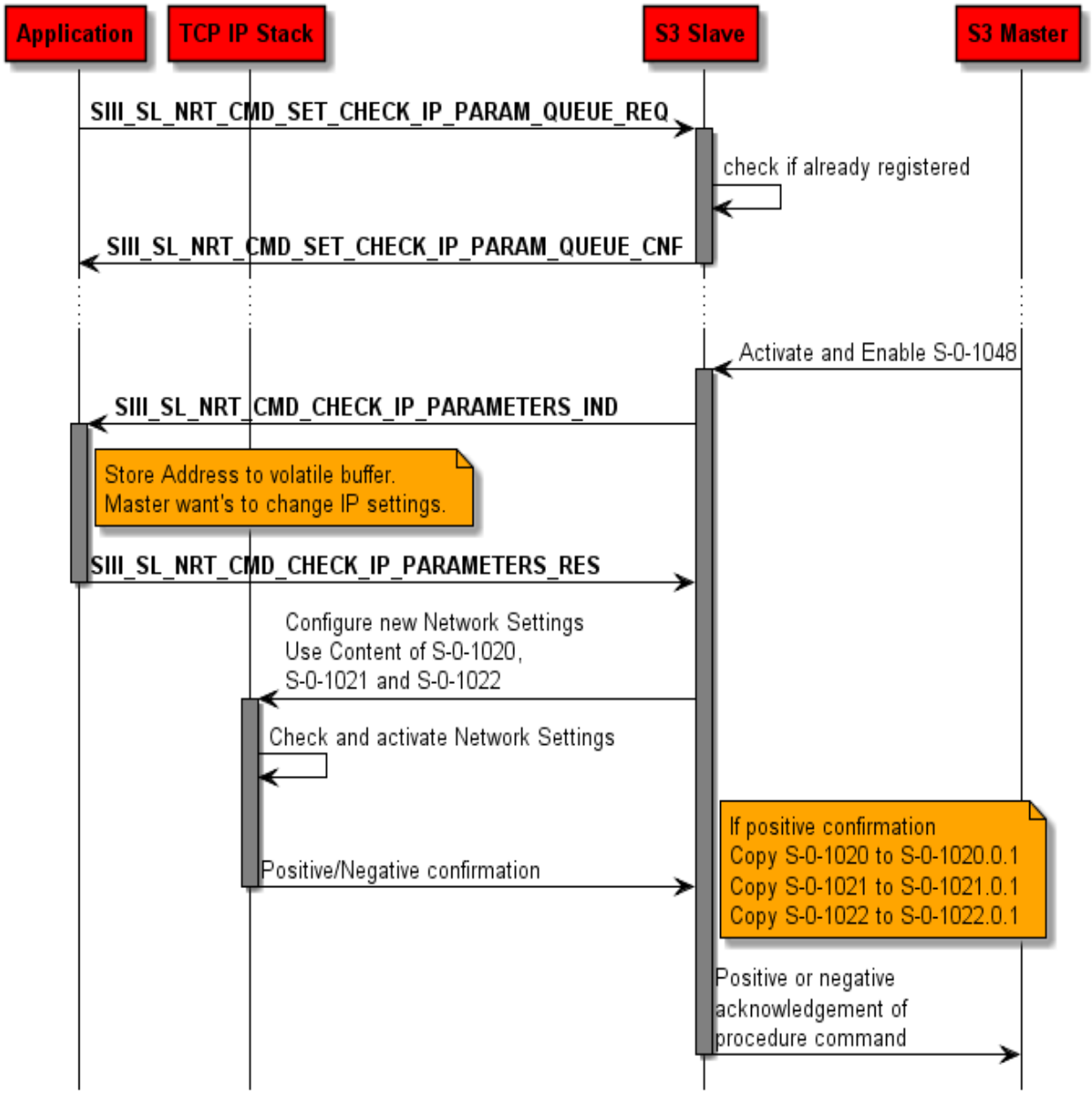


Figure 63: Register check IP parameter

The slave stack will send the new network settings each time if the master tries to activate them through S-0-1048 or the S/IP request to change IP settings.

Packet Structure Reference

```
typedef struct
{
  TLR_PACKET_HEADER_T   tHead;
} TLR_EMPTY_PACKET_T;
```

Packet Description

Structure TLR_EMPTY_PACKET_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x6690	SIII_SL_NRT_SET_CHECK_IP_PARAM_QUEUE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 78: `SIII_SL_NRT_SET_CHECK_IP_PARAM_QUEUE_REQ /CNF` – IDN Register Check IP Parameter Request Packet

Packet Structure Reference

```
typedef struct
{
  TLR_PACKET_HEADER_T   tHead;
} TLR_EMPTY_PACKET_T;
```

Packet Description

Structure TLR_EMPTY_PACKET_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x6691	SIII_SL_NRT_SET_CHECK_IP_PARAM_QUEUE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 79: SIII_SL_NRT_SET_CHECK_IP_PARAM_QUEUE_REQ /CNF – IDN Register Check IP Parameter Confirmation Packet

5.9.2 Check IP Parameters

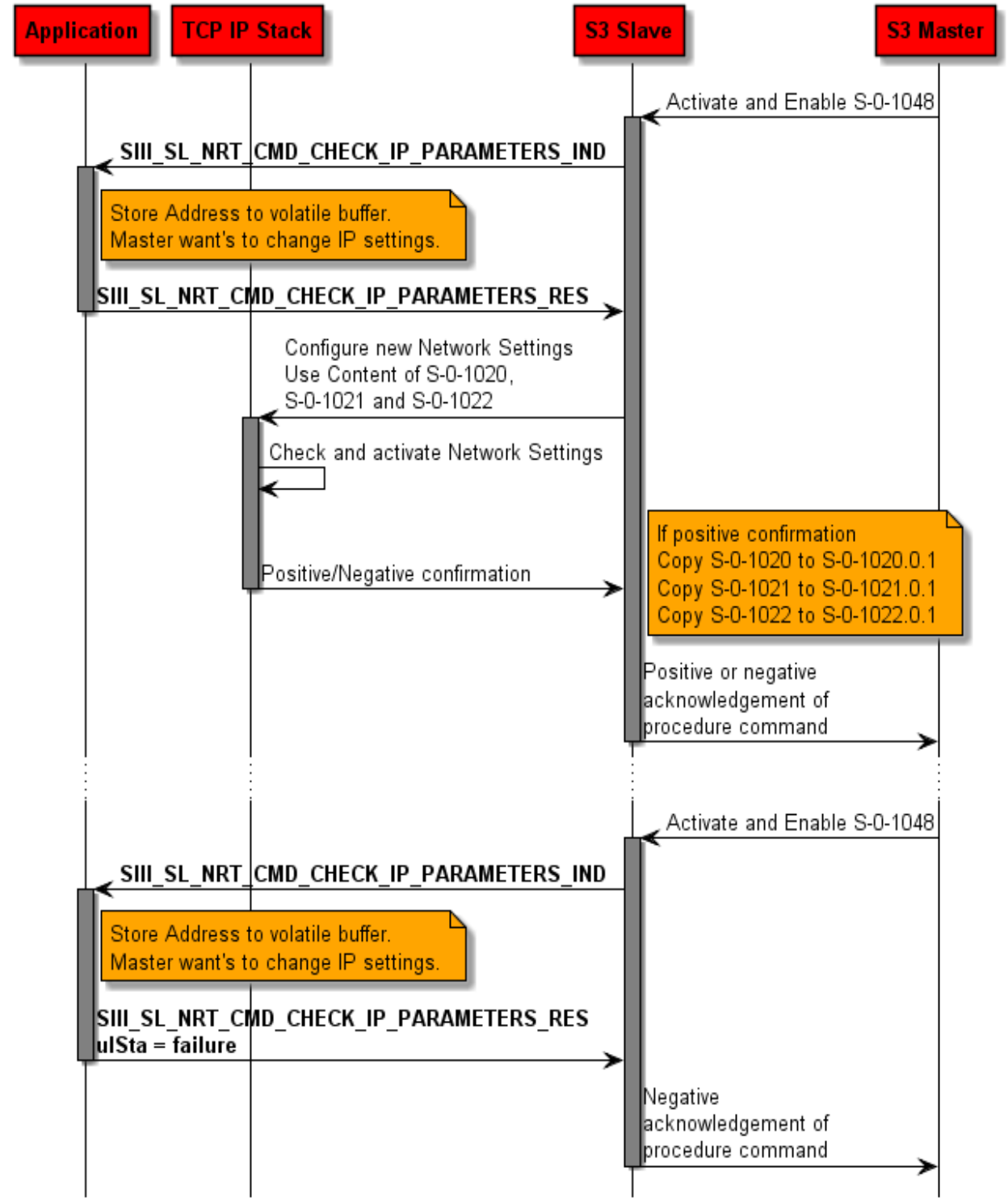


Figure 64: Check IP parameters

The packet Check IP Parameters indicates a change in the IP settings. To receive this packet, previously a Register Check IP Parameters Request has to be done. The indication contain the new IP settings that shall be activated.

Packet Structure Reference

```
/* if this flag is set, the TCP IP is configured with an IP address obtained through S-0-
1020 */
#define VAL_SIII_SL_NRT_CHECKIP_PARAM_TCP_CFG_FLAG_IP_ADDR            0x00000001
/* if this flag is set, the TCP IP is configured with an Subnet Mask obtained through S-
0-1021 */
#define VAL_SIII_SL_NRT_CHECKIP_PARAM_TCP_CFG_FLAG_NET_MASK           0x00000002
/* if this flag is set, the TCP IP is configured with an Gateway Address obtained through
S-0-1022 */
#define VAL_SIII_SL_NRT_CHECKIP_PARAM_TCP_CFG_FLAG_GATEWAY            0x00000004
/* This Flag is set through the IDN S-0-1048.0.1. If this flag is set, ulIpAddr,
ulNetMask and ulGateway will never be set. abHostname is used to send DHCP Option 81 */
#define VAL_SIII_SL_NRT_CHECKIP_PARAM_TCP_CFG_FLAG_DHCP               0x00000010
/* If this flag is set the IP settings shall be stored non volatile. Flag could just be
set if NRTPCv2 is enabled.*/
#define VAL_SIII_SL_NRT_CHECKIP_PARAM_TCP_CFG_STORE_DATA_NON_VOLATILE 0x01000000

typedef struct SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_DATA_Ttag
{
  /** Parameters for TCP/IP-Stack */
  TLR_UINT32      ulTcpFlags;
  /** IP Address (IP is little-endian formatted) */
  TLR_UINT32      ulIpAddr;
  /** Netmask (little endian formatted) */
  TLR_UINT32      ulNetMask;
  /** Gateway address (little endian formatted) */
  TLR_UINT32      ulGateway;
  TLR_UINT8       bHostNameLength;
  TLR_UINT8       abHostName[255];
} SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_DATA_T;

typedef struct SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_Ttag
{
  TLR_PACKET_HEADER_T                              tHead;
  SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_DATA_T      tData;
} SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_T;
```

Packet Description

Structure SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	17+n	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x6680	SIII_SL_NRT_CMD_CHECK_IP_PARAMETERS_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Structure SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_T			Type: Indication
tData - Structure SIII_SL_NRT_CHECK_IP_PARAMETERS_IND_DATA_T			
ulTcpFlags	UINT32	Bit mask	See “Packet Structure Reference” above
ulIpAddr	UINT32		The new IP address
ulNetMask	UINT32		The new network mask
ulGateway	UINT32		The new gateway address
bHostNameLength	UINT8	0…254	Length of the host-name
abHostName	UINT8[254]		The new host-name

Table 80: SIII_SL_NRT_CMD_CHECK_IP_PARAMETERS_IND/RES – NRT Check IP Parameters Indication Packet

This packet is returned from the application to the stack if the new IP settings shall be configured to the TCP/IP task. If the new settings may not be configured to TCP/IP task the confirmation must contain an `ulSta` value different than 0. If the configuration may be used `ulSta` must be 0.

Packet Structure Reference

```
typedef struct SIII_SL_NRT_CHECK_IP_PARAMETERS_RES_Ttag
{
  TLR_PACKET_HEADER_T                              tHead;
} SIII_SL_NRT_CHECK_IP_PARAMETERS_RES_T;
```

Packet Description

Structure SIII_SL_NRT_CHECK_IP_PARAMETERS_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x6681	SIII_SL_NRT_CMD_CHECK_IP_PARAMETERS_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 81: SIII_SL_NRT_CMD_CHECK_IP_PARAMETERS_IND/RES – NRT Check IP Parameters Response Packet

5.9.3 Activate Network Settings

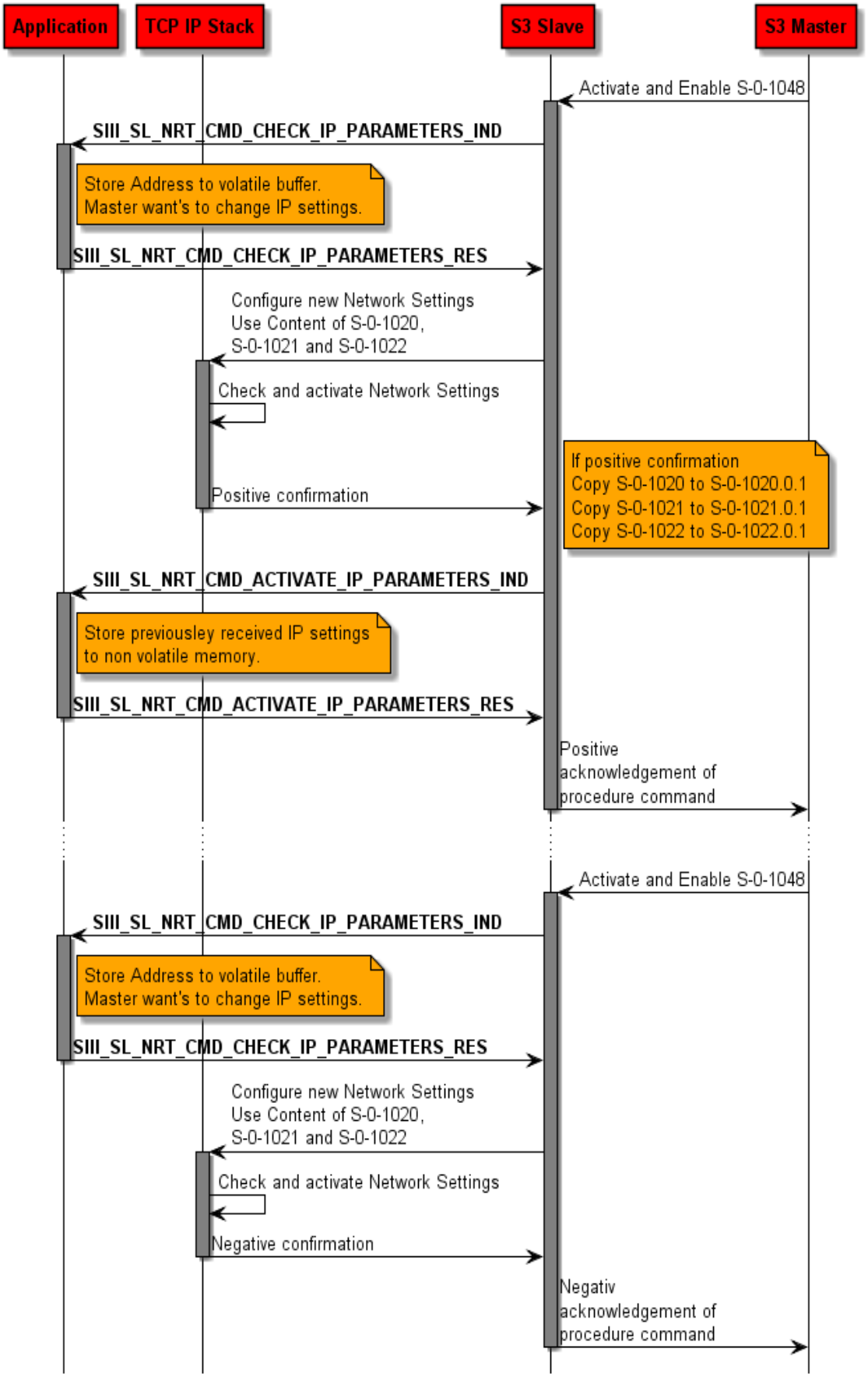


Figure 65: Activate Network settings

This indication is sent from the stack to the application if the new IP settings have been configured to the TCP/IP stack and the TCP/IP stack acknowledged the change of the new settings positively.

Packet Structure Reference

```
typedef struct
{
  TLR_PACKET_HEADER_T   tHead;
} TLR_EMPTY_PACKET_T;
```

Packet Description

Structure TLR_EMPTY_PACKET_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x6684	SIII_SL_NRT_CMD_ACTIVATE_IP_PARAMETERS_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 82: SIII_SL_NRT_CMD_ACTIVATE_IP_PARAMETERS_IND/RES – Activate IP Parameters Indication Packet

Packet Structure Reference

```
typedef struct
{
  TLR_PACKET_HEADER_T   tHead;
} TLR_EMPTY_PACKET_T;
```

Packet Description

Structure TLR_EMPTY_PACKET_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x6685	SIII_SL_NRT_CMD_ACTIVATE_IP_PARAMETERS_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 83: SIII_SL_NRT_CMD_ACTIVATE_IP_PARAMETERS_IND/RES – Activate IP Parameters Response Packet

5.10 Indications for registered Applications

To get some basic indications the application has to register via `RCX_REGISTER_APP_REQ/CNF`.

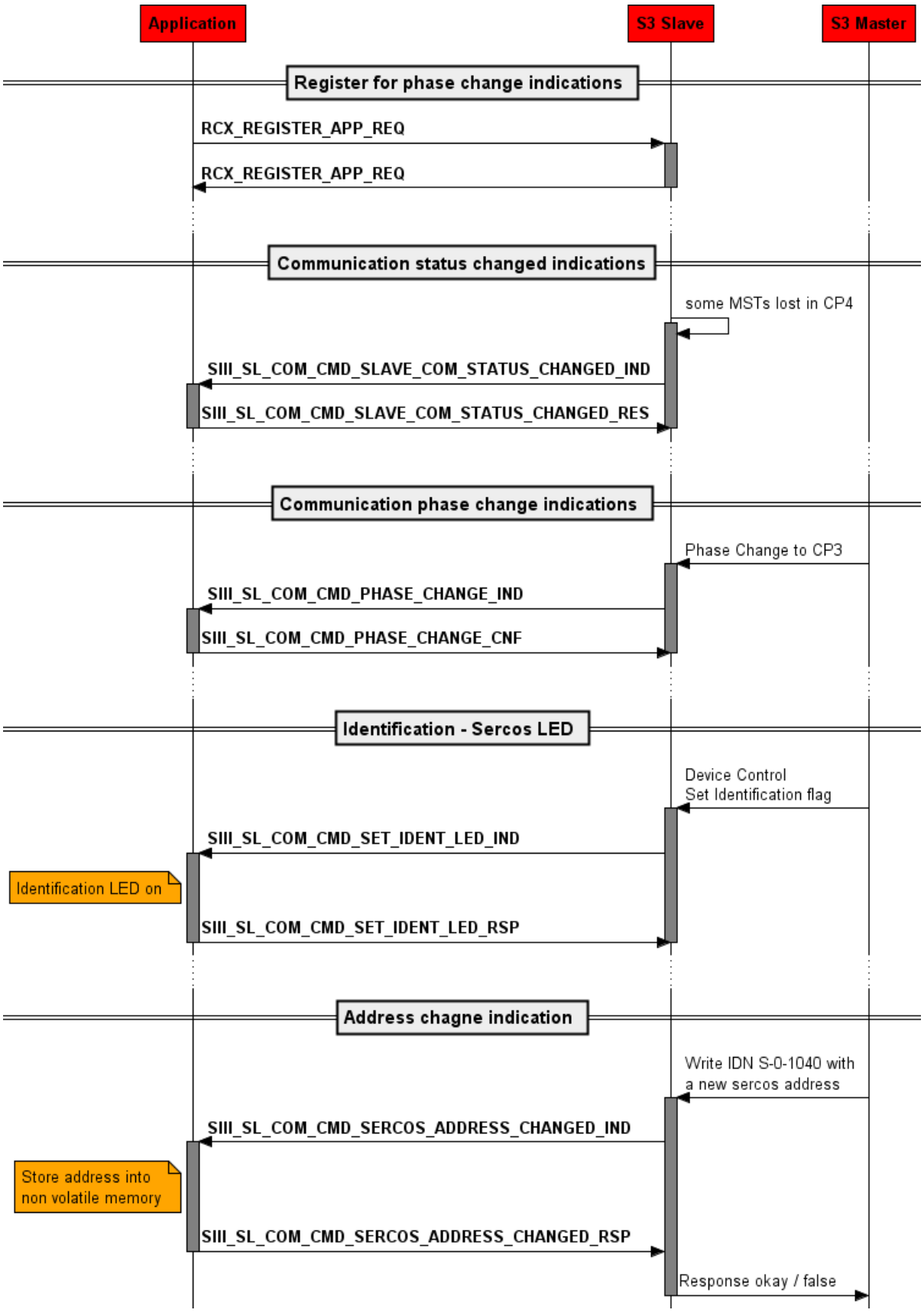


Figure 66: Packets which must be handled if application is registered

5.10.1 COM Status Changed Indication

The indication SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND occurs every time there is a change in the communication status, i. e. there is a new C1D Diagnostic, a communication error occurred, the loopback status has changed or a C2D Diagnostic is active.

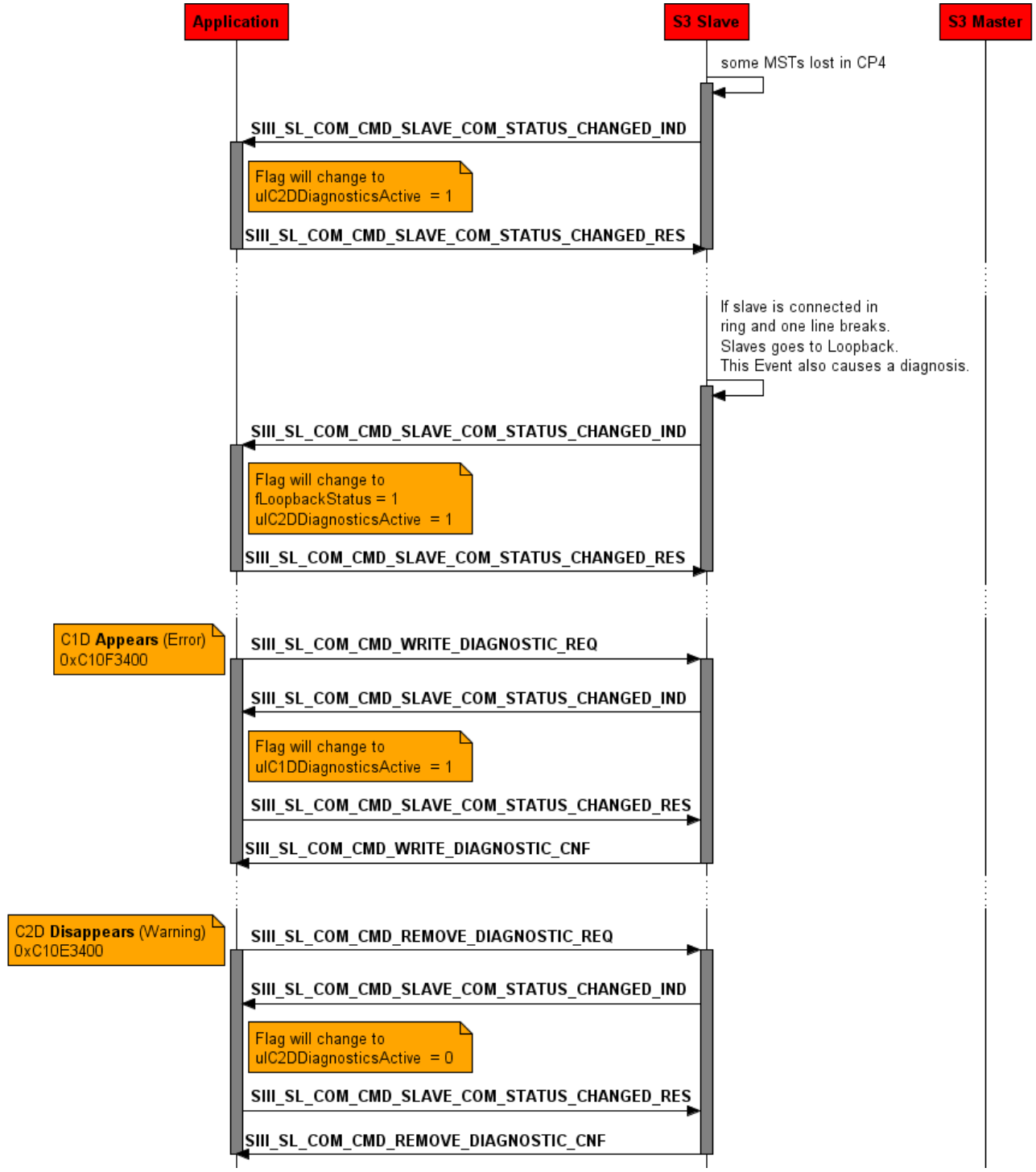


Figure 67: Communication status change indications for 1 slave device

Some of these parameters are bit fields. If you have configured just a single slave device you will receive a “1” or a zero. But if you configured a multiple slave device each bit position represents a slave. Bit 0 corresponds to slave 1, bit 1 to slave 2 etc.

Packet Structure Reference

```
typedef struct SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_DATA_Ttag
{
  TLR_UINT32                                        ulC1DDiagnosticsActive;
  TLR_UINT32                                        ulCommunicationErrorActive;
  TLR_BOOLEAN32                                     fLoopbackStatus;
  TLR_UINT32                                        ulC2DDiagnosticsActive;
} SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_DATA_T;

typedef struct SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_Ttag
{
  TLR_PACKET_HEADER_T                               tHead;
  SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_DATA_T   tData;
} SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_T;
```

Packet Description

Structure SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x326A	SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_IND_DATA_T			
ulC1DDiagnosticsActive	UINT32		Bit field for C1D Diagnostics Active
ulCommunicationErrorActive	UINT32		Bit field for Communication Error Active
fLoopbackStatus	BOOLEAN32		Current loopback status
ulC2DDiagnosticsActive	UINT32		Bit field for C2D Diagnostics Active

Table 84: SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_IND/RES – COM Status Changed Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_RES_Ttag
{
  TLR_PACKET_HEADER_T  tHead;
} SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_RES_T;
```

Packet Description

Structure SIII_SL_COM_SLAVE_COM_STATUS_CHANGED_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x326B	SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 85: SIII_SL_COM_CMD_SLAVE_COM_STATUS_CHANGED_RES – COM Status Changed Response Packet

5.10.2 Phase Change Indication

The indication SIII_SL_COM_CMD_PHASE_CHANGE_IND/RES occurs every time a change in the communication phase of the Sercos slave occurs. Phase change signaling is enabled via rcX packet RCX_REGISTER_APP_REQ.

For more information on the communication phases of the Sercos state machine see section *Communication Phases* on page 23 of this document.

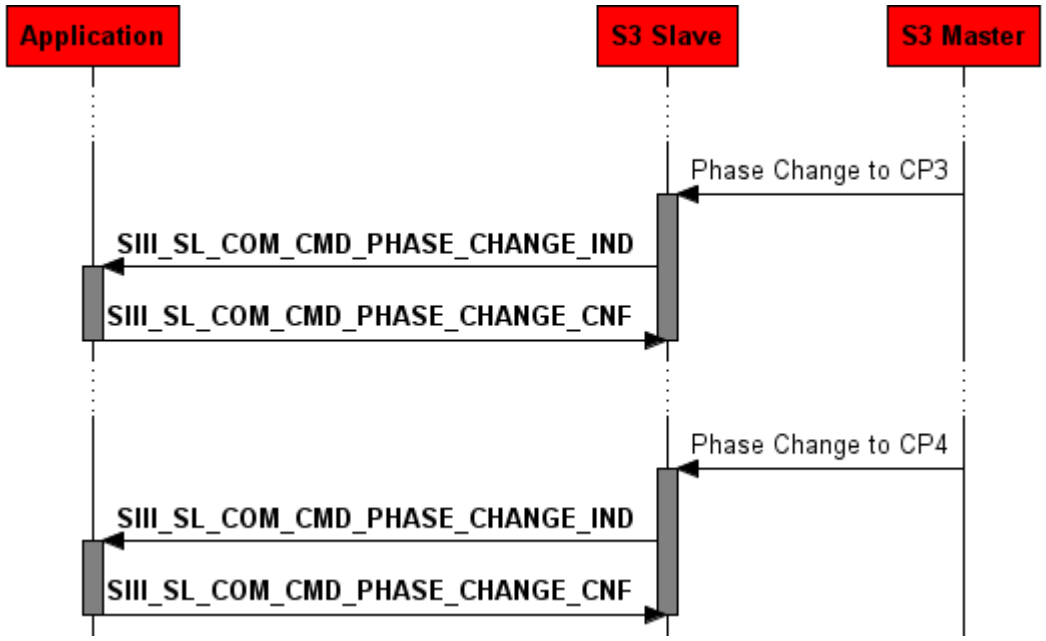


Figure 68: Phase change indication

Packet Structure Reference

```
typedef struct SIII_SL_COM_PHASE_CHANGE_IND_DATA_Ttag
{
  UINT8                                 bCurrentComPhase;
} SIII_SL_COM_PHASE_CHANGE_IND_DATA_T;

typedef struct SIII_SL_COM_PHASE_CHANGE_IND_Ttag
{
  TLR_PACKET_HEADER_T                   tHead;
  SIII_SL_COM_PHASE_CHANGE_IND_DATA_T   tData;
} SIII_SL_COM_PHASE_CHANGE_IND_T;
```

Packet Description

Structure SIII_SL_COM_PHASE_CHANGE_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	1	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3206	SIII_SL_COM_CMD_PHASE_CHANGE_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_PHASE_CHANGE_IND_DATA_T			
bCurrentComPhase	UINT8	0..4, 0x30…0x33, 0x7F	Current communication phase, see *Table 87: Possible communication phases (values of bCurrentComPhase)*.

Table 86: `SIII_SL_COM_PHASE_CHANGE_IND_T` – Phase Change Indication Packet

The current communication phase (i. e. the communication phase valid after the change) can be found in variable `bCurrentComPhase`. This packet applies to all slave instances at once.

Value	Symbolic Name	Communication phase
0x00	VAL_SIII_SL_COM_PHASE_CHANGE_IND_CP0	Communication Phase 0
0x01	VAL_SIII_SL_COM_PHASE_CHANGE_IND_CP1	Communication Phase 1
0x02	VAL_SIII_SL_COM_PHASE_CHANGE_IND_CP2	Communication Phase 2
0x03	VAL_SIII_SL_COM_PHASE_CHANGE_IND_CP3	Communication Phase 3
0x04	VAL_SIII_SL_COM_PHASE_CHANGE_IND_CP4	Communication Phase 4
0x7F	VAL_SIII_SL_COM_PHASE_CHANGE_IND_NRT	Non Real-Time Mode (NRT Mode)
0x30	VAL_SIII_SL_COM_PHASE_CHANGE_IND_HP0	Hotplug phase 0
0x31	VAL_SIII_SL_COM_PHASE_CHANGE_IND_HP1	Hotplug phase 1
0x32	VAL_SIII_SL_COM_PHASE_CHANGE_IND_HP2	Hotplug phase 2 (Phase equals to CP2, perform your CP2 initialization handling)
0x33	VAL_SIII_SL_COM_PHASE_CHANGE_IND_HP2_S_0_127	Hotplug phase 2 and Procedure Command S-0-0127 was executed. (Phase equals to CP3, perform your CP3 initialization handling)

Table 87: Possible communication phases (values of `bCurrentComPhase`)

Packet Structure Reference

```
typedef struct SIII_SL_COM_PHASE_CHANGE_RES_Ttag
{
  TLR_PACKET_HEADER_T  tHead;
} SIII_SL_COM_PHASE_CHANGE_RES_T;
```

Packet Description

Structure SIII_SL_COM_PHASE_CHANGE_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3207	SIII_SL_COM_CMD_PHASE_CHANGE_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing, do not change

Table 88: `SIII_SL_COM_PHASE_CHANGE_RES_T` – Phase Change Response Packet

5.10.3 Enable/Disable Identification LED

If you develop a LOM application you have to serve the Sercos S LED if this packet is received. LFW applications just have to send the response back to the stack.

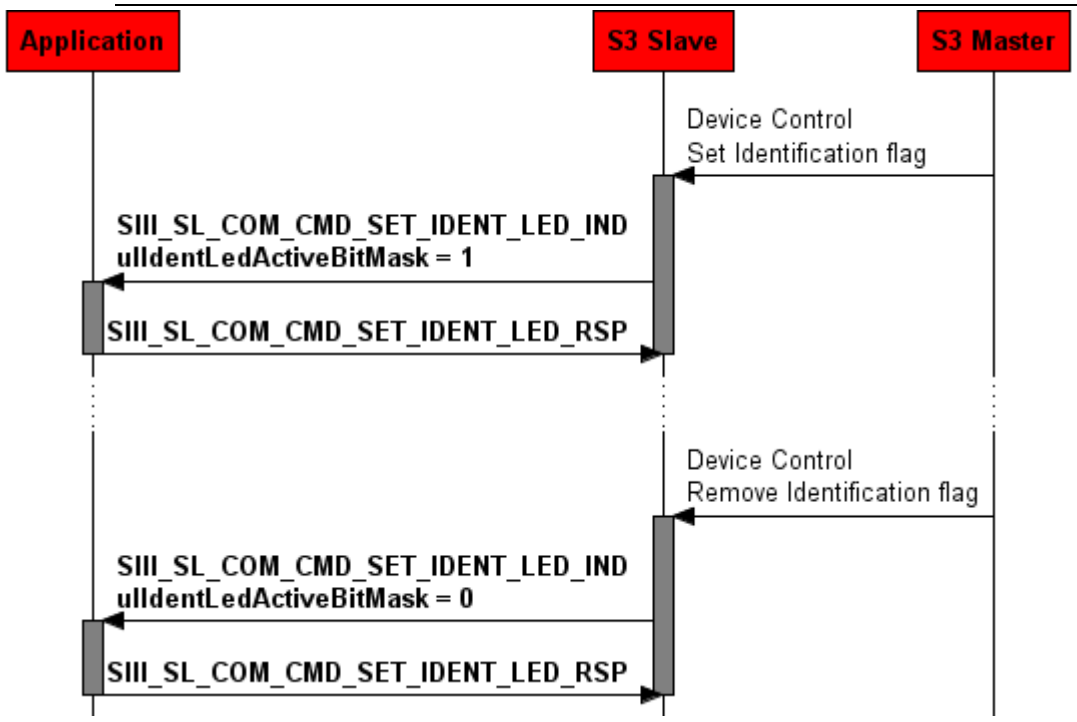


Figure 69: LFW Identification LED handling

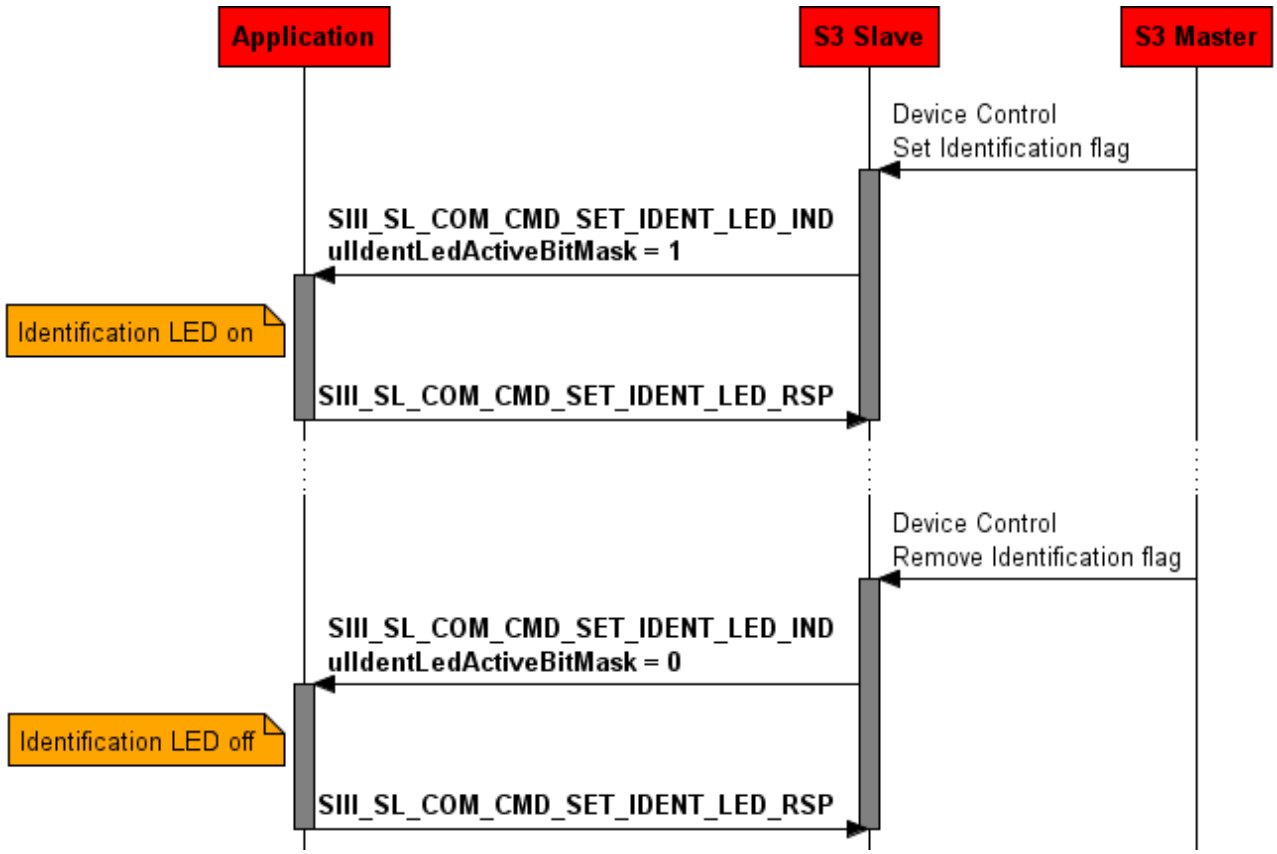


Figure 70: LOM Identification LED handling

Bit 0 describes for which `SlaveIdx` the indication LED shall be activated. But slaves just have to check if the Bit-Mask equals zero. There is just one LED for all slaves. If the Identification service is requested for at least one slave, the LED has to be activated.

This indication is enabled via the rcX packet `RCX_REGISTER_APP_REQ`.

Packet Structure Reference

```
typedef struct SIII_SL_COM_SET_IDENT_LED_IND_DATA_Ttag
{
  UINT32                                ulIdentLedActiveBitMask;
} SIII_SL_COM_SET_IDENT_LED_IND_DATA_T;

typedef struct SIII_SL_COM_SET_IDENT_LED_IND_Ttag
{
  TLR_PACKET_HEADER_T                   tHead;
  SIII_SL_COM_SET_IDENT_LED_IND_DATA_T  tData;
} SIII_SL_COM_SET_IDENT_LED_IND_T;
```

Packet Description

Structure SIII_SL_COM_SET_IDENT_LED_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3260	SIII_SL_COM_CMD_SET_IDENT_LED_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_SET_IDENT_LED_IND_DATA_T			
ulIdentLedActiveBitMask	UINT32	Bit mask	Active bit mask for the ident LED. If not zero, the LED must show the “Identification” pattern. If zero a lower priority state must be shown.

Table 89: `SIII_SL_COM_SET_IDENT_LED_IND_T` – Set Ident LED Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_SET_IDENT_LED_RES_Ttag
{
  TLR_PACKET_HEADER_T  tHead;
} SIII_SL_COM_SET_IDENT_LED_RES_T;
```

Packet Description

Structure SIII_SL_COM_SET_IDENT_LED_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3261	SIII_SL_COM_CMD_SET_IDENT_LED_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change

Table 90: SIII_SL_COM_SET_IDENT_LED_RES_T – Set Ident LED Response Packet

5.10.4 Sercos Address Changed Indication

The received address must be stored in a non-volatile memory. After power cycle the stack has to start up with the last configured Sercos address (use the address for set-configuration).

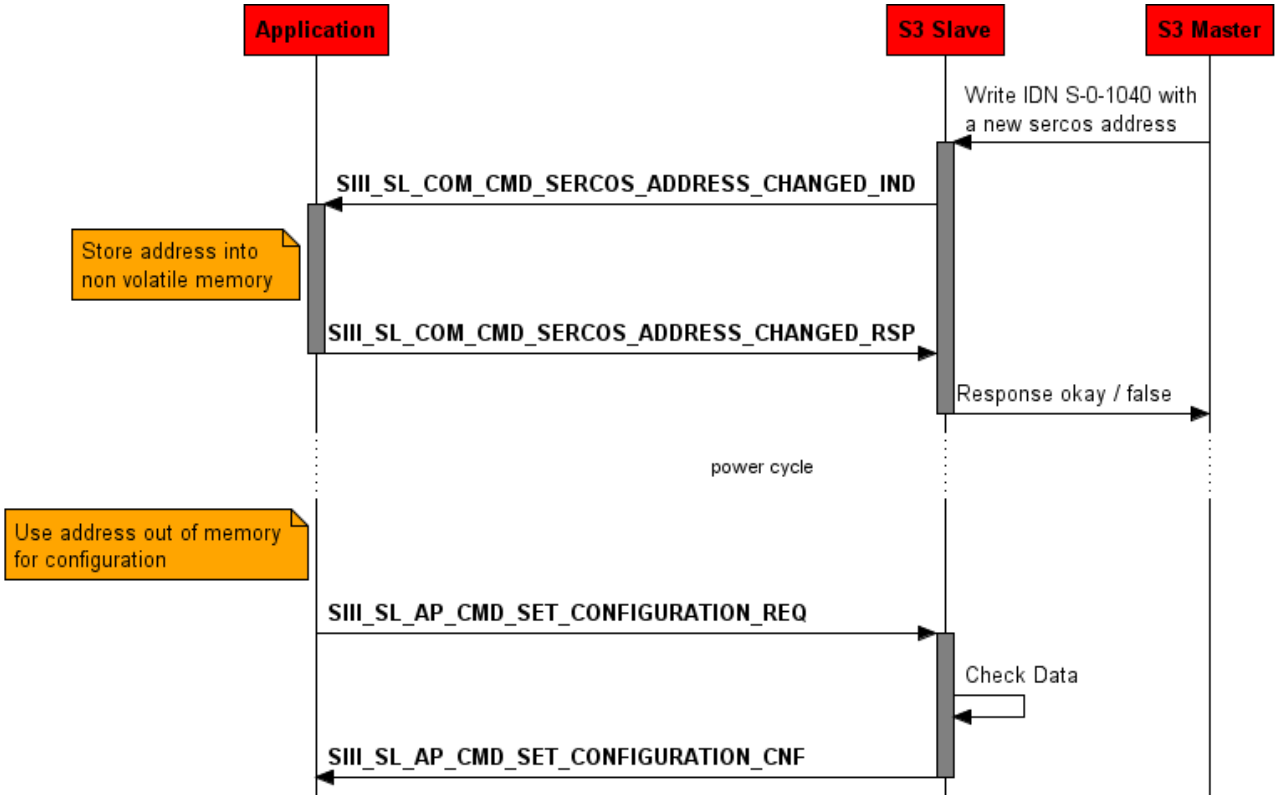


Figure 71: Address change indication

The indication packet `SIII_SL_COM_CMD_SERCOS_ADDRESS_CHANGED_IND` informs about the change of the SERCOS address. The new SERCOS address is delivered in the variable `usSERCOSAddress`. The indication is generated if the master changes the “IDN S-0-1040 SERCOS address”.

This indication is enabled via the rcX packet `RCX_REGISTER_APP_REQ`.

Packet Structure Reference

```
typedef struct SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_DATA_Ttag
{
  TLR_UINT8                                     bSlaveIdx;
  TLR_UINT16                                    usSERCOSAddress;
} SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_DATA_T;

typedef struct SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_DATA_T tData;
} SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_T;
```

Packet Description

Structure SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	3	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3268	SIII_SL_COM_CMD_SERCOS_ADDRESS_CHANGED_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device
usSERCOSAddress	UINT16	0..511	Sercos Address

Table 91: `SIII_SL_COM_SERCOS_ADDRESS_CHANGED_IND_T` – Set SERCOS Address changed Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_DATA_Ttag
{
  TLR_UINT8                                       bSlaveIdx;
} SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_DATA_T;

typedef struct SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_Ttag
{
  TLR_PACKET_HEADER_T                             tHead;
  SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_DATA_T   tData;
} SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_T;
```

Packet Description

Structure SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3269	SIII_SL_COM_CMD_SERCOS_ADDRESS_CHANGED_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change
tData - Structure SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_DATA_T			
bSlaveIdx	UINT8	0..7	Index of the slave device

Table 92: SIII_SL_COM_SERCOS_ADDRESS_CHANGED_RES_T – Set SERCOS Address changed Response Packet

5.11 Special Packets for Linkable Object Module

5.11.1 Initialization Completed Indication

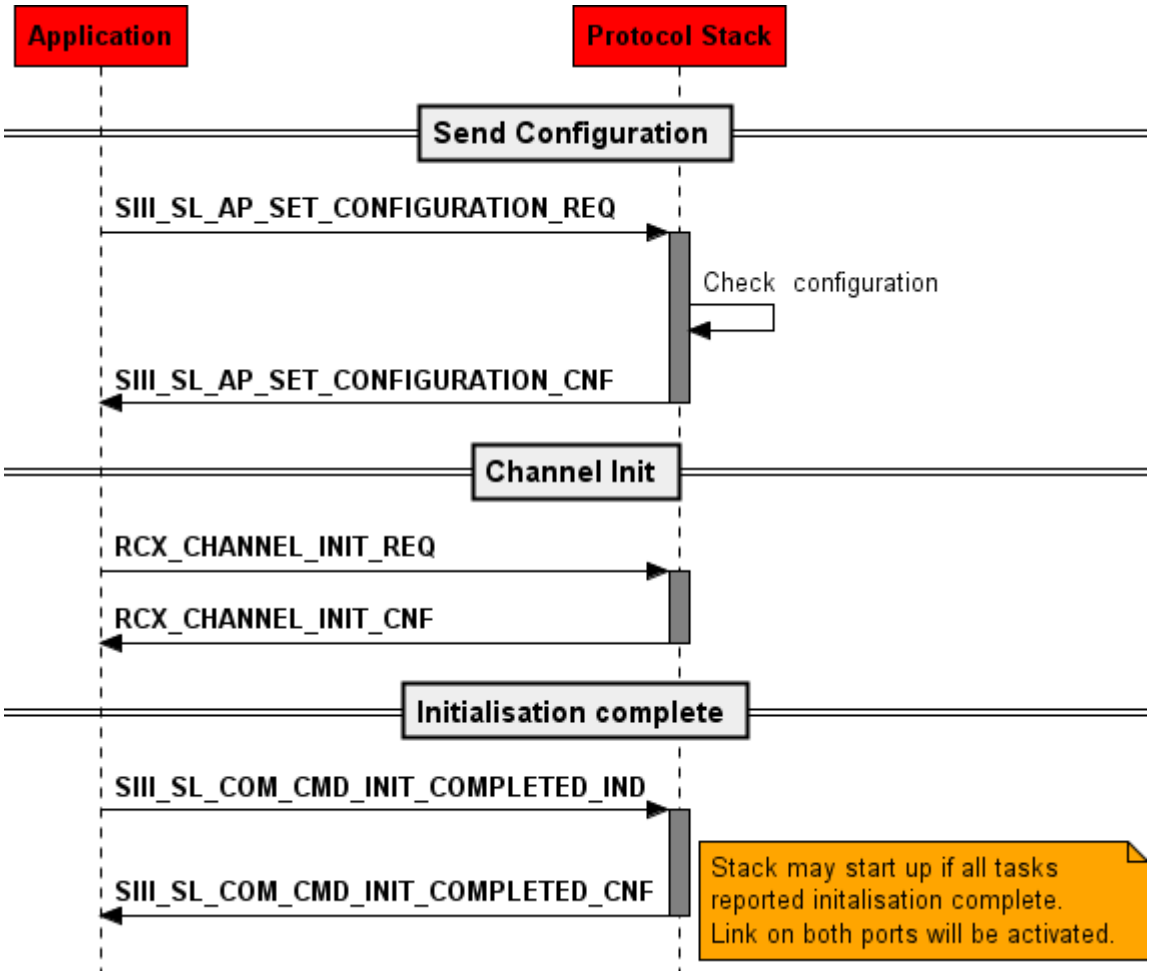


Figure 72: Initialization complete indication

If the application task has done everything necessary for initialization this packet shall be sent. If the slave stack has been configured via start-up parameters, it will just wait until it receives this packet.

This must be configured in the `config.c` file to the COM task. See the example below.

Example for start-up parameters:

```
static const SIII_SL_COM_STARTUPPARAMETER_T tSIII_SlCOM_Parameters =
{
  TLR_TASK_SERCOSIII_SL_COM, SIII_SL_COM_STARTUP_PARAM_VERSION_V3,
  FALSE,
  SIII_SL_COM_INIT_COMPLETED_SVC_TASK_READY | SIII_SL_COM_INIT_COMPLETED_APP_0_STARTED |
SIII_SL_COM_INIT_COMPLETED_NRT_TASK_READY
};
```

Bit	Description
31-20	Currently not used
19-16	SIII_SL_COM_INIT_COMPLETED_APP_3_STARTED SIII_SL_COM_INIT_COMPLETED_APP_2_STARTED SIII_SL_COM_INIT_COMPLETED_APP_1_STARTED SIII_SL_COM_INIT_COMPLETED_APP_0_STARTED 4 flags reserved for application
7-3	Currently not used
2	SIII_SL_COM_INIT_COMPLETED_NRT_TASK_STARTED 0 - NRT task of Sercos Slave not ready 1 - NRT task of Sercos Slave ready
1	SIII_SL_COM_INIT_COMPLETED_AP_TASK_STARTED 0 - AP task of Sercos Slave not ready 1 - AP task of Sercos Slave ready
0	SIII_SL_COM_INIT_COMPLETED_SVC_TASK_STARTED 0 - SVC task of Sercos Master not ready 1 - SVC task of Sercos Master ready

Table 93: Explanation of `ulInitCompletedFlags`

Packet Structure Reference

```
#define SIII_SL_COM_INIT_COMPLETED_SVC_TASK_STARTED           0x00000001
#define SIII_SL_COM_INIT_COMPLETED_AP_TASK_STARTED            0x00000002
#define SIII_SL_COM_INIT_COMPLETED_NRT_TASK_STARTED           0x00000004

/* application reserved bits 16-31 */
#define SIII_SL_COM_INIT_COMPLETED_APP_0_STARTED              0x00010000
#define SIII_SL_COM_INIT_COMPLETED_APP_1_STARTED              0x00020000
#define SIII_SL_COM_INIT_COMPLETED_APP_2_STARTED              0x00040000
#define SIII_SL_COM_INIT_COMPLETED_APP_3_STARTED              0x00080000

typedef struct SIII_SL_COM_INIT_COMPLETED_IND_DATA_Ttag
{
  TLR_UINT32                              ulInitCompletedFlags;
} SIII_SL_COM_INIT_COMPLETED_IND_DATA_T;

typedef struct SIII_SL_COM_INIT_COMPLETED_IND_Ttag
{
  TLR_PACKET_HEADER_T                     tHead;
  SIII_SL_COM_INIT_COMPLETED_IND_DATA_T   tData;
} SIII_SL_COM_INIT_COMPLETED_IND_T;
```

Packet Description

Structure SIII_SL_COM_INIT_COMPLETED_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3270	SIII_SL_COM_CMD_INIT_COMPLETED_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_INIT_COMPLETED_IND_DATA_T			
ulInitCompletedFlags	UINT32	Bit mask	Initialization completion flags

Table 94: `SIII_SL_COM_INIT_COMPLETED_IND_T` – Init Completed Indication Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_INIT_COMPLETED_RES_Ttag
{
  TLR_PACKET_HEADER_T  tHead;
} SIII_SL_COM_INIT_COMPLETED_RES_T;
```

Packet Description

Structure SIII_SL_COM_INIT_COMPLETED_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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, unchanged
ulSta	UINT32	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3271	SIII_SL_COM_CMD_INIT_COMPLETED_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change

Table 95: SIII_SL_COM_INIT_COMPLETED_RES_T – Init Completed Response Packet

5.11.2 Set Sercos Addresses Request

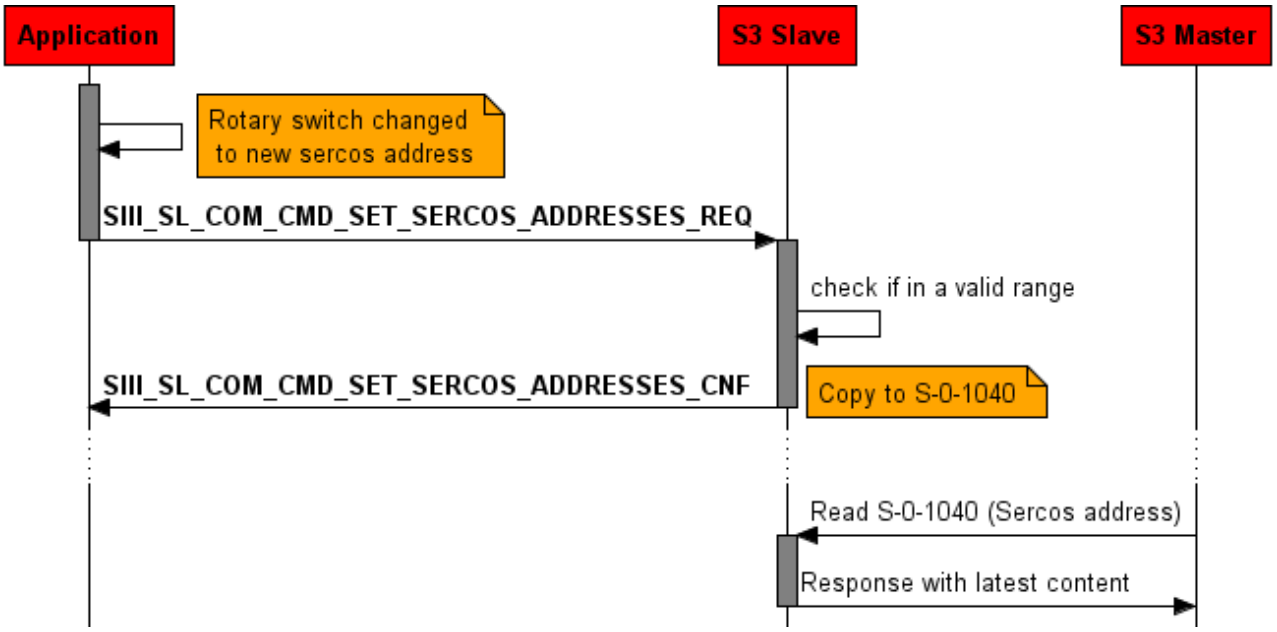


Figure 73: Set Sercos addresses

The packet `SIII_SL_COM_CMD_SET_SERCOS_ADDRESSES_REQ` changes the SERCOS addresses of the slave. A newly configured Sercos address is directly used in IDN S-0-1040. In CP0 the address is used for address allocation.

All of the addresses (up to 8) are configured by sending one single packet.

- `ulChangeFlags` is a bit list, which selects the addresses to be reconfigured.
- `ulAddressNotChangeableBits` is a bit list which defines whether the SERCOS Address is write protected during CP2. If the appropriate bit is set to 1, the master is unable to modify the address. If a rotary switch is connected, this bit must be set if the Sercos address is not zero. If the Sercos address is zero, the bit must be removed.
- `ausSERCOSAddresses[]` is an array of 8 SERCOS addresses.

Packet Structure Reference

```
typedef struct SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_DATA_Ttag
{
  TLR_UINT32                                  ulChangeFlags;
  TLR_UINT32                                  ulAddressNotChangeableBits;
/* addresses enabled by ulChangeFlags */
  TLR_UINT16                                  ausSERCOSAddresses[8];
} SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_DATA_T;

typedef struct SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_Ttag
{
  TLR_PACKET_HEADER_T                         tHead;
  SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_DATA_T  tData;
} SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_T;
```

Packet Description

Structure SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ QUE_S3_SL_ COM	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	24	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	0	See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3266	SIII_SL_COM_CMD_SET_SERCOS_ADDRESSES_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_DATA_T			
ulChangeFlags	UINT32	bit list	Change Flags
ulAddressNotChangeableBits	UINT32	bit list	Address not changeable bits
ausSERCOSAddresses[8]	UINT16[]	0..511	Array of Sercos addresses

Table 96: `SIII_SL_COM_SET_SERCOS_ADDRESSES_REQ_T` – Set SERCOS Addresses Request Packet

Packet Structure Reference

```
typedef struct SIII_SL_COM_SET_SERCOS_ADDRESSES_CNF_Ttag
{
  TLR_PACKET_HEADER_T  tHead;
} SIII_SL_COM_SET_SERCOS_ADDRESSES_CNF_T;
```

Packet Description

Structure SIII_SL_COM_SET_SERCOS_ADDRESSES_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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, unchanged
ulSta	UINT32		See section *Status/Error Codes Overview*
ulCmd	UINT32	0x3267	SIII_SL_COM_CMD_SET_SERCOS_ADDRESSES_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change

Table 97: SIII_SL_COM_SET_SERCOS_ADDRESSES_CNF_T – Set SERCOS Addresses Confirmation Packet

5.11.3 Register for Trace buffer update indications

This packet is needed to register for trace buffer update indications. Trace buffer update indications are needed to handle the Sercos LED by application. Also if the application wants to serve additional trace buffers the indication is needed. Each time, the stack inserts a diagnosis message into its trace buffer this indication will be sent to the application.

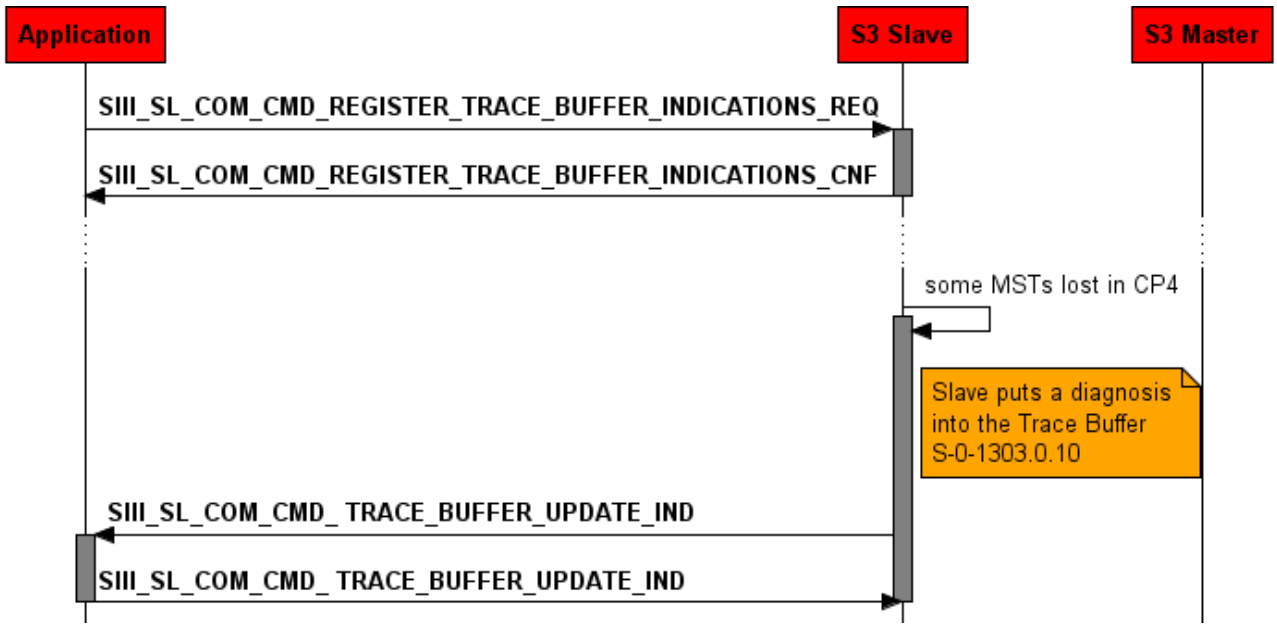


Figure 74: Register trace buffer update indications

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct
SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
  TLR_UINT8                                          bAllInserts; /* if set to 1 also
own registered indications are send back */
} __TLR_PACKED_POST SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_DATA_T;

typedef struct
{
  TLR_PACKET_HEADER_T   tHead;
SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_DATA_T    tData;
} SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32A6	SIII_SL_COM_CMD_REGISTER_TRACE_BUFFER_INDICATIONS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	The slave index.
bAllInserts	UINT8	bit list	0 = just indications if a diagnosis is reported not from this task 1 = also indications if this task reports a diagnosis

Table 98: SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_T /CNF – IDN Register Trace buffer update indications

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct
SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_CNF_DATA_Ttag
{
  TLR_UINT8                                              bSlaveIdx;
} __TLR_PACKED_POST SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_CNF_DATA_T;

typedef struct
{
  TLR_PACKET_HEADER_T   tHead;
SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_CNF_DATA_T    tData;
} SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_CNF_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32A7	SIII_SL_COM_CMD_REGISTER_TRACE_BUFFER_INDICATIONS_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_CNF_DATA_T			
bSlaveIdx	UINT8	0…7	The slave index.

Table 99: SIII_SL_COM_REGISTER_TRACE_BUFFER_INDICATIONS_REQ_T /CNF – IDN Register Trace buffer update confirmation

5.11.3.1 Trace buffer update indications

Trace buffer update indications are needed to handle the Sercos LED by application. Also if the application want's to serve additional trace buffers the indication is needed. Each time, the stack inserts a diagnosis message into its trace buffer this indication will be send to the application.

To receive this packet a special register packet must be send to the stack (see previous chapter).

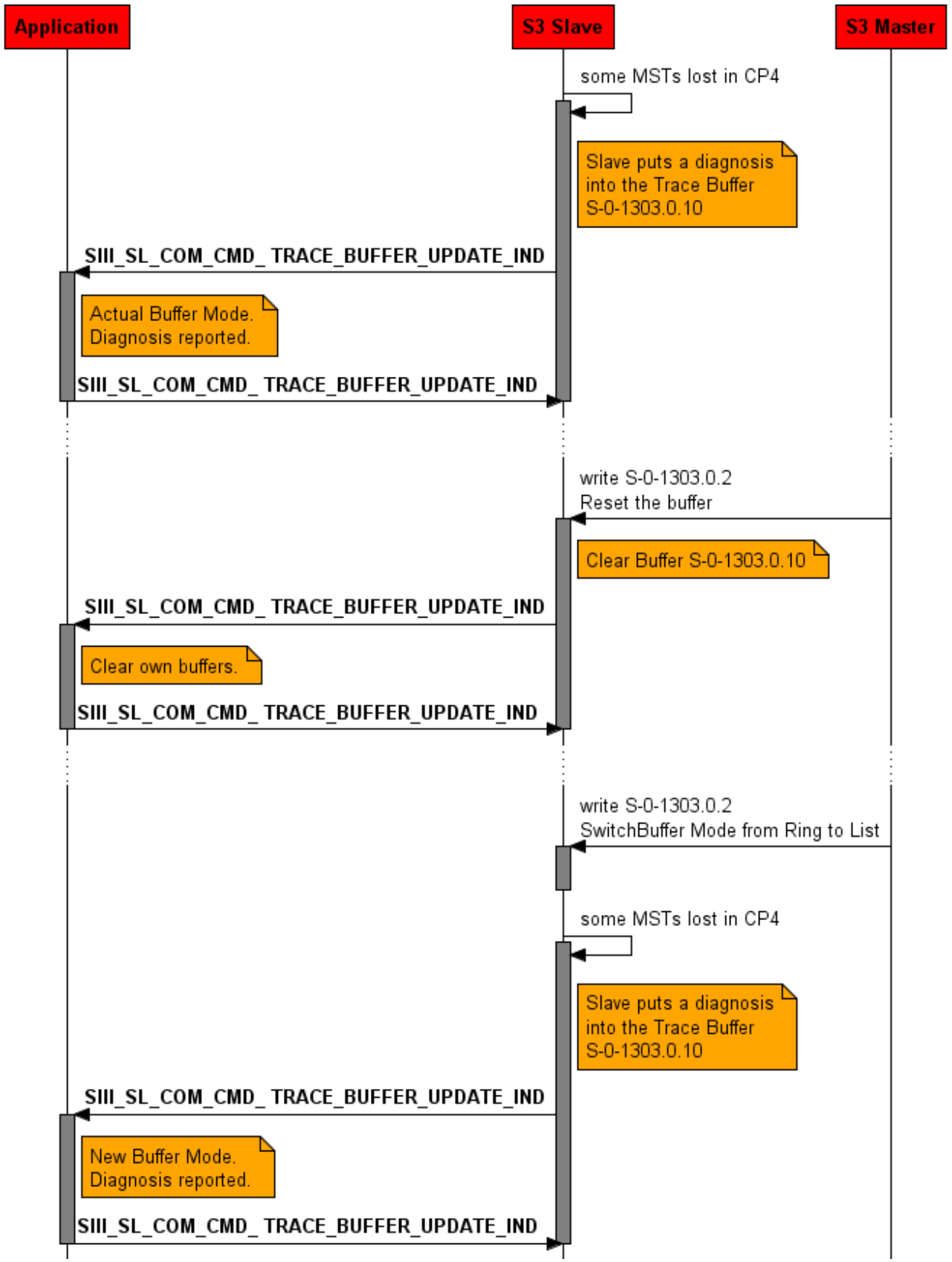


Figure 75: Trace buffer update indications

Packet Structure Reference

```
/* A new element was inserted into the Buffer. Also the user application has to add an
element into its own buffer */
#define SIII_SL_COM_PROC_CMD_TRACE_BUFFER_UPDATE_INSERTED_INTO_BUFFER    0
/* The buffer was reset, also the user application has to reset it's buffer */
#define SIII_SL_COM_PROC_CMD_TRACE_BUFFER_UPDATE_RESET_BUFFER            1

typedef __TLR_PACKED_PRE struct SIII_SL_COM_TRACE_BUFFER_UPDATE_IND_DATA_Ttag
{
  TLR_UINT8                                              bSlaveIdx;
  TLR_UINT8                                              bFlag;
  TLR_UINT32                                             ulDiagnosticNumber; /* just valid
if bFlag is set to zero; the currently inserted diagnostic number, should not be needed
by application task, just for information purpose */
  TLR_UINT16                                             usBufferMode;  /* 0 = Ringbuffer,
1 = Listbuffer */
} __TLR_PACKED_POST SIII_SL_COM_TRACE_BUFFER_UPDATE_IND_DATA_T;

typedef struct SIII_SL_COM_TRACE_BUFFER_UPDATE_IND_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
  SIII_SL_COM_TRACE_BUFFER_UPDATE_IND_DATA_T             tData;
} SIII_SL_COM_TRACE_BUFFER_UPDATE_IND_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_TRACE_BUFFER_UPDATE_IND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32A A	SIII_SL_COM_CMD_ TRACE_BUFFER_UPDATE_IND- Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_ TRACE_BUFFER_UPDATE_IND_DATA_T			
bSlaveIdx	UINT8	0..7	The slave index.
bFlag	UINT8	bit list	0 = a new diagnosis was added 1 = the master performed a reset of the trace buffer
ulDiagnosticNumber	UINT32		The diagnosis recently added to the trace buffer. Just valid if bFlag is set to zero.
usBufferMode	UINT16	0..1	The actual buffer Mode (may be changed during runtime through S-0-1303.0.2). 0 = ring buffer 1 = list buffer

Table 100: SIII_SL_COM_ TRACE_BUFFER_UPDATE_IND _T /CNF – IDN Trace buffer update indication

Packet Structure Reference

```
/* confirmation packet */
typedef struct SIII_SL_TRACE_BUFFER_UPDATE_RES_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
} SIII_SL_COM_TRACE_BUFFER_UPDATE_RES_T;
```

Packet Description

Structure SIII_SL_COM_ TRACE_BUFFER_UPDATE_RES_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32AB	SIII_SL_COM_CMD_ TRACE_BUFFER_UPDATE_RSP - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 101: SIII_SL_COM_ TRACE_BUFFER_UPDATE_RSP_T /CNF – IDN Trace buffer update response

5.11.4 Register for Test IDN diagnostic indications

This packet registers for diagnosis if S-0-1399.0.1 is written. The IDN is just for tests and needed by the Sercos conformizer. This packet is just required if the Application implements own trace buffers (S-0-1303.0.12) and traces e.g. the IDN S-0-1500.x.32.

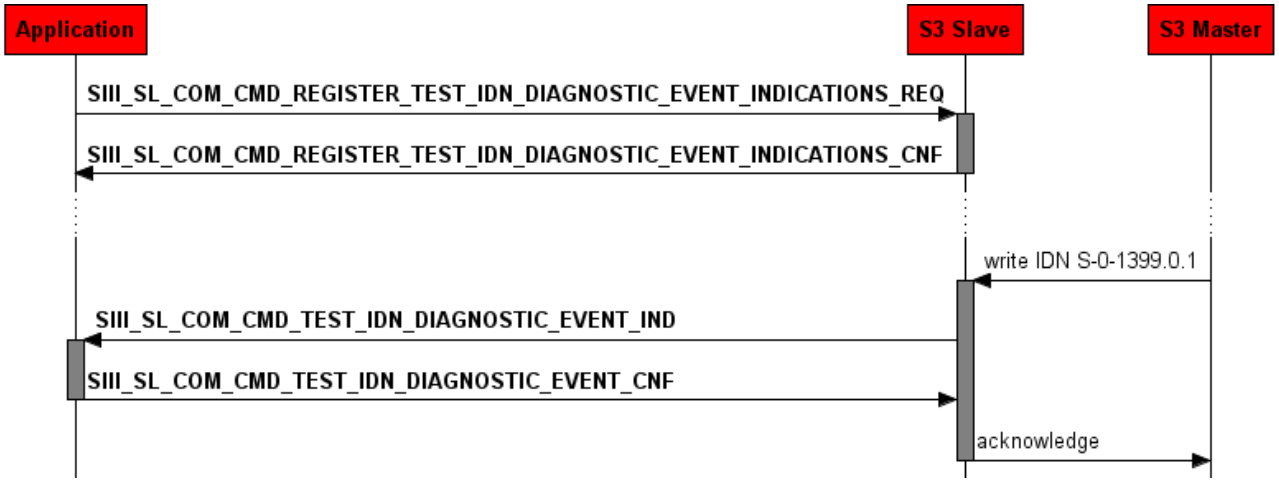


Figure 76: Register for test IDN diagnostic indication

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct
SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_DATA_Ttag
{
  TLR_UINT8                                              bSlaveIdx;
} __TLR_PACKED_POST
SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_DATA_T;

typedef struct SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
  SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_DATA_T    tData;
} SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_T;
```

Packet Description

Structure SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32AE	SIII_SL_COM_CMD_REGISTER_TEST_IDN_DIAGNOSTIC_ EVENT_INDICATIONS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_REQ_DATA_T			
bSlaveIdx	UINT8	0..7	The slave index.

Table 102: `SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATION_REQ/CNF` – Register for Test IDN diagnostic indication

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct
SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_DATA_Ttag
{
  TLR_UINT8                                              bSlaveIdx;
} __TLR_PACKED_POST
SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_DATA_T;

typedef struct SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
  SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_DATA_T    tData;
} SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_T;
```

Packet Description

Structure **SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_T**			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32A7	SIII_SL_COM_CMD_REGISTER_TEST_IDN_DIAGNOSTIC_ EVENT_INDICATIONS_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATIONS_CNF_DATA_T			
bSlaveIdx	UINT8	0…7	The slave index.

Table 103: SIII_SL_COM_REGISTER_TEST_IDN_DIAGNOSTIC_EVENT_INDICATION_REQ/CNF – Register for Test IDN diagnostic confirmation

5.11.4.1 Test IDN diagnostic indications

These indications are needed to handle the LED pattern if the Test IDN (S-0-1399.0.1) is written with a diagnosis. Also if the application wants to implement additional own trace buffers this indication must be handled.

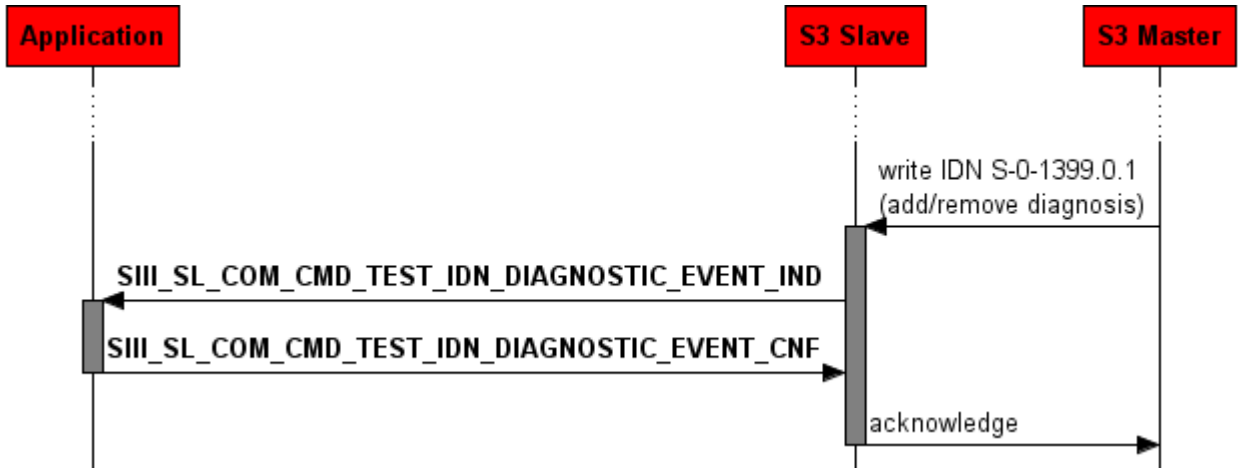


Figure 77: Test IDN diagnostic indication

Packet Structure Reference

```
typedef __TLR_PACKED_PRE struct SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_DATA_Ttag
{
  TLR_UINT8                                          bSlaveIdx;
  TLR_BOOLEAN8                                       bBufferMode; /* 0 = Ringbuffer, 1
= Listbuffer */
  TLR_UINT32                                         ulDiagnosticNumber;
} __TLR_PACKED_POST SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_DATA_T;

typedef struct SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_Ttag
{
  TLR_PACKET_HEADER_T                                tHead;
  SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_DATA_T  tData;
} SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_T;
```

Packet Description

Structure SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32B2	SIII_SL_COM_CMD_TEST_IDN_DIAGNOSTIC_EVENT_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_DATA_T			
bSlaveIdx	UINT8	0..7	The slave index.
usBufferMode	UINT16	0..1	The actual buffer Mode (may be changed during runtime through S-0-1303.0.2). 0 = ring buffer 1 = list buffer
ulDiagnosticNumber	UINT32		The diagnosis recently added to the trace buffer through S-0-1399.0.1

Table 104: SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND/RSP – Test IDN diagnostic indication

Packet Structure Reference

```
/* confirmation packet */
typedef __TLR_PACKED_PRE struct TEST_IDN_DIAGNOSTIC_UPDATE_RES_DATA_Ttag
{
  TLR_UINT8                                              bSlaveIdx;
} __TLR_PACKED_POST TEST_IDN_DIAGNOSTIC_UPDATE_RES_DATA_T;

/* confirmation packet */
typedef struct SIII_SL_TEST_IDN_DIAGNOSTIC_UPDATE_RES_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
  TEST_IDN_DIAGNOSTIC_UPDATE_RES_DATA_T                  tData;
} SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_RES_T;
```

Packet Description

Structure SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_RES_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	0	See section Status/Error Codes Overview
ulCmd	UINT32	0x32B3	SIII_SL_COM_CMD_TEST_IDN_DIAGNOSTIC_EVENT_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
tData - Structure SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND_DATA_T			
ulDiagnosticNumber	UINT32		The diagnosis recently added to the trace buffer through S-0-1399.0.1

Table 105: SIII_SL_COM_TEST_IDN_DIAGNOSTIC_UPDATE_IND/RSP – Test IDN diagnostic response

6 Topics for programmers using Linkable Object Modules

6.1 Accessing the Protocol Stack by Programming the AP Task's Queue

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

Getting the Receiver Task Handle of the Process Queue

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

ASCII Queue name	Description
"QUE_S3_SL_AP"	Name of the Sercos Device AP-Task process queue
"QUE_S3_SL_COM"	Name of the Sercos Device COM-Task process queue
"QUE_S3_SL_IDN"	Name of the Sercos slave IDN-Task process queue
"QUE_S3_SL_NRT	Name of the Sercos slave NRT-Task process queue
QUE_S3_SL_SVC	Name of the Sercos slave SVC-Task process queue
QUE_SIII_SIP	Name of the Sercos slave SIP-Task process queue

Table 106: Names of Queues in Sercos Slave Firmware

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

7 Status/Error Codes Overview

7.1 Status/Error Codes AP-Task

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok.
0xC0510001	TLR_E_SERCOSIII_SL_AP_INVALID_S3S_NXD Invalid database for Sercos slave.

Table 107: Status/Error Codes of the Sercos Slave AP-Task

7.2 Status/Error Codes IDN-Task

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC0850001	TLR_E_SERCOSIII_SL_IDN_COMMAND_INVALID Invalid command.
0xC0850004	TLR_E_SERCOSIII_SL_IDN_ALREADY_EXISTS IDN already exists.
0xC0850005	TLR_E_SERCOSIII_SL_IDN_ATTRIBUTE_INVALID Invalid attribute specified.
0xC0850006	TLR_E_SERCOSIII_SL_IDN_INVALID_MAX_DATA_SIZE_SPECIFIED Invalid max data size specified.
0xC0850007	TLR_E_SERCOSIII_SL_IDN_SLAVE_INDEX_INVALID Slave index invalid.
0xC0850008	TLR_E_SERCOSIII_SL_IDN_UNDEFINED_NOTIFY_ALREADY_IN_USE Undefined notify already in use.
0xC0850009	TLR_E_SERCOSIII_SL_IDN_INVALID_ELEMENT_ID Invalid element id.
0xC085000A	TLR_E_SERCOSIII_SL_IDN_APP_PACKET_RESPONSE_INVALID Application's Response Packet invalid.
0xC085000B	TLR_E_SERCOSIII_SL_IDN_APP_TRANSFER_TOO_LONG Application's Transfer Data too long.
0xC085000C	TLR_E_SERCOSIII_SL_IDN_APP_TRANSFER_LENGTH_WRONG Application's Transfer Data length is invalid.
0xC085000D	TLR_E_SERCOSIII_SL_IDN_APP_MTU_TOO_LOW Application's MTU is too low.
0xC085000E	TLR_E_SERCOSIII_SL_IDN_INVALID_DEST_ID Invalid DestId.
0xC085000F	TLR_E_SERCOSIII_SL_IDN_LISTS_CANNOT_HAVE_A_MINIMUM_VALUE Lists cannot have a minimum value.
0xC0850010	TLR_E_SERCOSIII_SL_IDN_LISTS_CANNOT_HAVE_A_MAXIMUM_VALUE Lists cannot have a maximum value.
0xC0850011	TLR_E_SERCOSIII_SL_IDN_NAME_EXCEEDS_ALLOCATED_LENGTH Name exceeds allocated length.
0xC0850012	TLR_E_SERCOSIII_SL_IDN_UNIT_EXCEEDS_ALLOCATED_LENGTH Unit exceeds allocated length.
0xC0850013	TLR_E_SERCOSIII_SL_IDN_OPDATA_EXCEEDS_ALLOCATED_LENGTH OpData exceeds allocated length.
0xC0850014	TLR_E_SERCOSIII_SL_IDN_INVALID_MAX_LIST_LENGTH Invalid max list length.

Hex Value	Definition / Description
0xC0850015	TLR_E_SERCOSIII_SL_IDN_DEFAULT_VALUE_EXCEEDS_ALLOCATED_LENGTH Default value exceeds allocated length.
0xC0850016	TLR_E_SERCOSIII_SL_IDN_MINIMUM_AND_MAXIMUM_VALUE_MUST_BE_USED_TOGETHER Minimum and maximum value must be used together.
0xC0850017	TLR_E_SERCOSIII_SL_IDN_USER_APPLICATION_TRANSFER_ERROR User application transfer error.
0xC0850018	TLR_E_SERCOSIII_SL_IDN_INTERNALLY_HANDLED_IDN IDN is internally handled.
0xC0851001	TLR_E_SERCOSIII_SL_IDN_NO_IDN IDN not available.
0xC0851009	TLR_E_SERCOSIII_SL_IDN_INVALID_ACCESS_TO_ELEMENT_1 Invalid access to element 1.
0xC0852001	TLR_E_SERCOSIII_SL_IDN_NO_NAME No Name.
0xC0852002	TLR_E_SERCOSIII_SL_IDN_NAME_TRANSMISSION_IS_TOO_SHORT Name transmission is too short.
0xC0852003	TLR_E_SERCOSIII_SL_IDN_NAME_TRANSMISSION_IS_TOO_LONG Name transmission is too long.
0xC0852004	TLR_E_SERCOSIII_SL_IDN_NAME_CANNOT_BE_CHANGED Name cannot be changed (read only).
0xC0852005	TLR_E_SERCOSIII_SL_IDN_NAME_IS_WRITE_PROTECTED_AT_THIS_TIME Name is write protected at this time.
0xC0852016	TLR_E_SERCOSIII_SL_IDN_NAME_APPLICATION_TIMEOUT Application timeout occurs in the slave.
0xC0853002	TLR_E_SERCOSIII_SL_IDN_ATTRIBUTE_TRANSMISSION_IS_TOO_SHORT Attribute transmission is too short.
0xC0853003	TLR_E_SERCOSIII_SL_IDN_ATTRIBUTE_TRANSMISSION_IS_TOO_LONG Attribute transmission is too long.
0xC0853004	TLR_E_SERCOSIII_SL_IDN_ATTRIBUTE_CANNOT_BE_CHANGED Attribute cannot be changed (read only).
0xC0853005	TLR_E_SERCOSIII_SL_IDN_ATTRIBUTE_IS_WRITE_PROTECTED_AT_THIS_TIME Attribute is write protected at this time.
0xC0853016	TLR_E_SERCOSIII_SL_IDN_ATTRIBUTE_APPLICATION_TIMEOUT Application timeout occurs in the slave.
0xC0854001	TLR_E_SERCOSIII_SL_IDN_NO_UNIT No unit.
0xC0854002	TLR_E_SERCOSIII_SL_IDN_UNIT_TRANSMISSION_IS_TOO_SHORT Unit transmission is too short.
0xC0854003	TLR_E_SERCOSIII_SL_IDN_UNIT_TRANSMISSION_IS_TOO_LONG Unit transmission is too long.
0xC0854004	TLR_E_SERCOSIII_SL_IDN_UNIT_CANNOT_BE_CHANGED Unit cannot be changed (read only).
0xC0854005	TLR_E_SERCOSIII_SL_IDN_UNIT_IS_WRITE_PROTECTED_AT_THIS_TIME Unit is write protected at this time.
0xC0854016	TLR_E_SERCOSIII_SL_IDN_UNIT_APPLICATION_TIMEOUT Application timeout occurs in the slave.
0xC0855001	TLR_E_SERCOSIII_SL_IDN_NO_MINIMUM_VALUE No minimum value.
0xC0855002	TLR_E_SERCOSIII_SL_IDN_MINIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT Minimum value transmission is too short.
0xC0855003	TLR_E_SERCOSIII_SL_IDN_MINIMUM_VALUE_TRANSMISSION_IS_TOO_LONG Minimum value transmission is too long.
0xC0855004	TLR_E_SERCOSIII_SL_IDN_MINIMUM_VALUE_CANNOT_BE_CHANGED Minimum value cannot be changed (read only).

Hex Value	Definition / Description
0xC0855005	TLR_E_SERCOSIII_SL_IDN_MINIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME Minimum value is write protected at this time.
0xC0855016	TLR_E_SERCOSIII_SL_IDN_MINIMUM_VALUE_APPLICATION_TIMEOUT Application timeout occurs in the slave.
0xC0856001	TLR_E_SERCOSIII_SL_IDN_NO_MAXIMUM_VALUE No maximum value.
0xC0856002	TLR_E_SERCOSIII_SL_IDN_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT Maximum value transmission is too short.
0xC0856003	TLR_E_SERCOSIII_SL_IDN_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_LONG Maximum value transmission is too long.
0xC0856004	TLR_E_SERCOSIII_SL_IDN_MAXIMUM_VALUE_CANNOT_BE_CHANGED Maximum value cannot be changed (read only).
0xC0856005	TLR_E_SERCOSIII_SL_IDN_MAXIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME Maximum value is write protected at this time.
0xC0856016	TLR_E_SERCOSIII_SL_IDN_MAXIMUM_VALUE_APPLICATION_TIMEOUT Application timeout occurs in the slave.
0xC0857002	TLR_E_SERCOSIII_SL_IDN_OPDATA_TRANSMISSION_IS_TOO_SHORT Operation data transmission is too short.
0xC0857003	TLR_E_SERCOSIII_SL_IDN_OPDATA_TRANSMISSION_IS_TOO_LONG Operation data transmission is too long.
0xC0857004	TLR_E_SERCOSIII_SL_IDN_OPDATA_CANNOT_BE_CHANGED Operation data cannot be changed (read only).
0xC0857005	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_WRITE_PROTECTED_AT_THIS_TIME Operation data is write protected at this time.
0xC0857006	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_LOWER_THAN_MINIMUM_VALUE Operation data is lower than Minimum value.
0xC0857007	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_HIGHER_THAN_MAXIMUM_VALUE Operation data is higher than Maximum value.
0xC0857008	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_INVALID Invalid operation data.
0xC0857009	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_WRITE_PROTECTED_BY_PASSWORD Operation data is write protected by password.
0xC085700A	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_WRITE_PROTECTED_DUE_CYCLICALLY_CONFIGURED Operation data is write protected. It is configured cyclically.
0xC085700B	TLR_E_SERCOSIII_SL_IDN_OPDATA_INVALID_INDIRECT_ADDRESSING Invalid indirect addressing.
0xC085700C	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_WRITE_PROTECTED_DUE_OTHER_SETTINGS Operation data is write protected due other settings.
0xC085700D	TLR_E_SERCOSIII_SL_IDN_OPDATA_INVALID_FLOATING_POINT_NUMBER Invalid floating point number.
0xC085700E	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_WRITE_PROTECTED_AT_PARAMETERIZATION_LEVEL Operation data is write protected at parameterization level.
0xC085700F	TLR_E_SERCOSIII_SL_IDN_OPDATA_IS_WRITE_PROTECTED_AT_OPERATION_LEVEL Operation data is write protected at operation level.
0xC0857010	TLR_E_SERCOSIII_SL_IDN_OPDATA_PROCEDURE_COMMAND_ALREADY_ACTIVE Procedure command already active.
0xC0857011	TLR_E_SERCOSIII_SL_IDN_OPDATA_PROCEDURE_COMMAND_NOT_INTERRUPTIBLE Procedure command not interruptible.
0xC0857012	TLR_E_SERCOSIII_SL_IDN_OPDATA_PROCEDURE_COMMAND_NOT_EXECUTABLE_AT_THIS_TIME Procedure Command is not executable at this time (e.g. wrong slave state).
0xC0857013	TLR_E_SERCOSIII_SL_IDN_OPDATA_PROCEDURE_COMMAND_NOT_EXECUTABLE_INVALID_PARAM Procedure Command is not executable due invalid parameters.

Hex Value	Definition / Description
0xC0857014	TLR_E_SERCOSIII_SL_IDN_OPDATA_LIST_PARAMETER_NOT_EXCEPTED The received current length of list parameter does not match to expectation.
0xC0857015	TLR_E_SERCOSIII_SL_IDN_OPDATA_NOT_YET_CREATED_COMPLETELEY Operation data is not yet created completely (it takes more time to create the operation data, try it again later).
0xC0857016	TLR_E_SERCOSIII_SL_IDN_OPDATA_APPLICATION_TIMEOUT Application timeout occurs in the slave.

Table 108: Status/Error Codes of the Sercos Slave IDN-Task

7.3 Status/Error Codes COM-Task

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok.
0xC04E0031	TLR_E_SERCOSIII_SL_COM_INVALID_NUM_OF_SLAVES Invalid number of slaves or invalid slave number
0xC04E0032	TLR_E_SERCOSIII_SL_COM_INVALID_SYNC_VERSION Invalid or not supported SCP_Sync version.
0xC04E0033	TLR_E_SERCOSIII_SL_COM_INVALID_SMP_VERSION Invalid or not supported SCP_SMP version.
0xC04E0034	TLR_E_SERCOSIII_SL_COM_INVALID_NRT_VERSION Invalid or not supported SCP_NRT version.Invalid or not supported Sync configuration flags.
0xC04E0035	TLR_E_SERCOSIII_SL_COM_INVALID_SYNC_CONFIG_FLAGS. Invalid or not supported Sync configuration flags.
0xC04E0036	TLR_E_SERCOSIII_SL_COM_CONCLK_SIGNAL_TOO_SHORT ConClk is enabled, but signal length below minimum 1000 ns
0xC04E0037	TLR_E_SERCOSIII_SL_COM_INVALID_DIVCLK_SIGNAL DivClk is enabled, but signal length below minimum 1us or greater than 20us.
0xC04E0038	TLR_E_SERCOSIII_SL_COM_INVALID_DIVCLK_CONFIG DivClk is enabled, but config is invalid.
0xC04E0039	TLR_E_SERCOSIII_SL_COM_INVALID_SERCOS_ADR Invalid SERCOS address.
0xC04E003A	TLR_E_SERCOSIII_SL_COM_INVALID_SCP_VERSION Invalid or not supported SCP version.
0xC04E003B	TLR_E_SERCOSIII_SL_COM_INVALID_SLAVE_FLAGS Invalid or not supported slave flags.
0xC04E003C	TLR_E_SERCOSIII_SL_COM_DEFAULT_OD_NOT_AVAILABLE Default object dictionary not available.
0xC04E003D	TLR_E_SERCOSIII_SL_COM_INVALID_USER_SCP_VERSION Invalid or not supported USER_SCP version.
0xC04E003E	TLR_E_SERCOSIII_SL_COM_INVALID_FSP_TYPE Invalid or not supported FSP Type.
0xC04E003F	TLR_E_SERCOSIII_SL_COM_INVALID_PROCESS_DATA_IMG Invalid or not supported process data image, it is too large.
0xC04E0040	TLR_E_SERCOSIII_SL_COM_CLEAN_UP_OD_FAILED Cleaning up the object dictionary failed.
0xC04E0041	TLR_E_SERCOSIII_SL_COM_CONCLK_SIGNAL_TOO_LONG ConClk is enabled, but signal length above maximum 655350 ns.
0xC04E0042	TLR_E_SERCOSIII_SL_COM_RTDATA_LENGTH_NOT_EVEN The RT-Data length may not be odd. Configure a even length.
0xC04E0043	TLR_E_SERCOSIII_SL_COM_INVALID_SYSTEM_START_VALUE The value for `ulSystemStart` is out of range.
0xC04E0044	TLR_E_SERCOSIII_SL_COM_INVALID_VENDOR_CODE_VALUE The value for usVendorCode is out of range.
0xC04E0045	TLR_E_SERCOSIII_SL_COM_DUBLICATE_CONFIGURED_SERCOS_ADDRESSES At the multiple slave device one Sercos address was double configured.
0xC04E0046	TLR_E_SERCOSIII_SL_COM_INVALID_SIP_VERSION Invalid version of SIP.
0xC04E0047	TLR_E_SERCOSIII_SL_COM_RESET_ONLY_THROUGH_S_0_0099 A Reset of a C1D error code is just allowed with S-0-0099.
0xC04E0048	TLR_E_SERCOSIII_SL_COM_MINIMUM_VALUE_NOT_READABLE The minimum value is not readable.

Hex Value	Definition / Description
0xC04E0049	TLR_E_SERCOSIII_SL_COM_MAXIMUM_VALUE_NOT_READABLE The maximum value is not readable.
0xC04E004A	TLR_E_SERCOSIII_SL_COM_NAME_NOT_READABLE The name is not readable.
0xC04E004B	TLR_E_SERCOSIII_SL_COM_ATTRIBUTE_NOT_READABLE The attribute is not readable.
0xC04E004C	TLR_E_SERCOSIII_SL_COM_UNIT_NOT_READABLE The unit is not readable.
0xC04E004D	TLR_E_SERCOSIII_SL_COM_NO_ENTRY_DELETED No entry found to deleted.
0xC04E004E	TLR_E_SERCOSIII_SL_COM_MAXIMUM_REGISTRATION_LISTENERS_REACHED All listening slots are in use.
0xC04E004F	TLR_E_SERCOSIII_SL_COM_NO_DIAGNOSIS_NUMBER_SET No diagnosis number set. A diagnosis number must be different to zero.
0xC04E0050L	TLR_E_SERCOSIII_SL_COM_JUST_ALLOWED_IN_CP3_OR_CP4 Reconfiguration with this packet is just allowed in CP3 or CP4.
0xC04E0051L	TLR_E_SERCOSIII_SL_COM_JUST_ALLOWED_IN_CP2 Reconfiguration with this packet is just allowed in CP2.
0xC04E0052L	TLR_E_SERCOSIII_SL_COM_NO_VAR_CONFIG This packet is just allowed if VarCfg is configured.
0xC04E0053L	TLR_E_SERCOSIII_SL_COM_NO_STORAGE_LEFT No storage left.
0xC04E0054L	TLR_E_SERCOSIII_SL_COM_NUMBER_CONNECTIONS_EXCEEDED The number of connections exceeded.
0xC04E0055L	TLR_E_SERCOSIII_SL_COM_RX_BUFFER_INTERRUPT_WITHOUT_SYNC The configuration of the RX Buffer Interrupt is just possible if Sync is also enabled.
0xC04E0056L	TLR_E_SERCOSIII_SL_COM_RX_WRONG_SERCOS_VERSION_CONFIGURED The configured Sercos Version is invalid.

Table 109: Status/Error Codes of the Sercos Slave COM-Task

7.4 Status/Error Codes NRT-Task

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok.
0xC0360001	TLR_E_SERCOSIII_ETH_COMMAND_INVALID Invalid data in request detected.
0xC0360002	TLR_E_SIII_SL_NRT_INVALID_STARTUP_PARAMETER Invalid startup parameters.
0xC0360003	TLR_E_SIII_SL_NRT_LLD_NOT_STARTED Link Layer Driver not started.

Table 110: Status/Error Codes of the Sercos Slave NRT-Task

7.5 Status/Error Codes Ethernet Interface

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok.
0xC05D0001	TLR_E_ETH_INTF_COMMAND_INVALID Invalid command received.
0xC05D0002	TLR_E_ETH_INTF_CONFIG_LOCK Configuration is locked.
0xC05D0003	TLR_E_ETH_INTF_INVALID_PACKET_LENGHT Invalid packet length.
0xC05D0004	TLR_E_ETH_INTF_INVALID_MODE Invalid mode in request.
0xC05D0005	TLR_E_ETH_INTF_PARAM_AUTO_NEGOTIATION_PORT_0 Invalid parameter for auto-negotiation port 0.
0xC05D0006	TLR_E_ETH_INTF_PARAM_AUTO_NEGOTIATION_PORT_1 Invalid parameter for auto-negotiation port 1.
0xC05D0007	TLR_E_ETH_INTF_PARAM_DUPLEX_MODE_PORT_0 Invalid parameter for duplex mode port 0.
0xC05D0008	TLR_E_ETH_INTF_PARAM_DUPLEX_MODE_PORT_1 Invalid parameter for duplex mode port 1.
0xC05D0009	TLR_E_ETH_INTF_PARAM_TRANSMISSION_RATE_PORT_0 Invalid parameter for transmission rate port 0.
0xC05D000A	TLR_E_ETH_INTF_PARAM_TRANSMISSION_RATE_PORT_1 Invalid parameter for transmission rate port 1.
0xC05D000B	TLR_E_ETH_INTF_PARAM_AUTO_CROSSOVER_PORT_0 Invalid parameter for auto cross-over port 0.
0xC05D000C	TLR_E_ETH_INTF_PARAM_AUTO_CROSSOVER_PORT_1 Invalid parameter for auto cross-over port 1.
0xC05D000D	TLR_E_ETH_INTF_NO_CONFIGURATION Task is not configured.
0xC05D000E	TLR_E_ETH_INTF_APP_NOT_REGISTERED No application registered.
0xC05D000F	TLR_E_ETH_INTF_APP_SET_NOT_READY Application set not ready.
0xC05D0010L	TLR_E_ETH_INTF_LINK_DOWN No Ethernet link.
0xC05D0011	TLR_E_ETH_INTF_GET_SEND_BUFFER Failed to get send buffer.
0xC05D0012	TLR_E_ETH_INTF_SEND_FRAME Failed to send Ethernet frame.
0xC05D0013	TLR_E_ETH_INTF_SET_DRV_EDD_CFG Failed to set driver EDD configuration.
0xC05D0014	TLR_E_ETH_INTF_INVALID_ETH_PORT Invalid parameter for Ethernet port.
0xC05DFFFF	TLR_E_ETH_INTF_UNKNOWN_ERROR Unknown error detected.

Table 111: Status/Error Codes of the Ethernet Interface-Task

7.6 Status/Error Codes TFTP-Task

Hex Value	Definition / Description
0x00000000	TLR_S_OK Status ok.
0xC0B80001	TLR_E_TFTP_COMMAND_INVALID Invalid command received.
0xC0B80002	TLR_E_TFTP_INVALID_PARAMETER Invalid parameter received.
0xC0B80003	TLR_E_TFTP_STACK_BUSY The stack is currently operating.
0xC0B80004	TLR_E_TFTP_DISK_FULL Not enough space.
0xC0B80005	TLR_E_TFTP_INTERNAL_ERROR Some internal error occured.
0xC0B80006	TLR_E_TFTP_REMOTE_STATION_TIMEOUT Remote station error - Timeout.
0xC0B80007	TLR_E_TFTP_REMOTE_STATION_FILE_NOT_FOUND Remote station error - File Not Found.
0xC0B80008	TLR_E_TFTP_REMOTE_STATION_ACCESS_VIOLATION Remote station error - Access Violation.
0xC0B80009	TLR_E_TFTP_REMOTE_STATION_DISK_FULL Remote station error - Disk Full.
0xC0B8000A	TLR_E_TFTP_REMOTE_STATION_FILE_EXISTS Remote station error - File Exists.
0xC0B8000B	TLR_E_TFTP_REMOTE_STATION_NO_SUCH_USER Remote station error - No Such User.
0xC0B8000C	TLR_E_TFTP_REMOTE_STATION_UNEXPCTD_ERR Remote station error - Unexpected Error.
0xC0B8000D	TLR_E_TFTP_ROOT_PATH_NOT_EXIST The root path specified does not exist.
0xC0B8000E	TLR_E_TFTP_FIRMWARE_FILE_CORRUPT The firmware file checksum error.

Table 112: Status/Error Codes of the TFTP –Task

7.7 Vendor-specific Sercos Error Codes

The following vendor-specific Sercos error codes may be transmitted to the master via the service channel:

0xCnnn vendor-specific Sercos error codes	
0xC000	unknown internal error
0xC001	internal error in IDN task
0xC002	out of memory
0xC003	packet out of memory
0xC004	packet done failed
0xC005	send packet failed
0xC006	get packet failed
0xC007	release packet failed
0xC008	error during user application transfer e.g. application response not within 5 seconds

Table 113: Vendor-specific Sercos Error Codes

7.8 Other Status Codes

7.8.1 Drive Status Code „operational state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x00	A	0010	Drive HALT is active
0x00	A	0012	Control and power sections ready for operation
0x00	A	0013	Ready for power on
0x00	A	0100	Torque control
0x00	A	0101	Velocity control
0x00	A	0102	Position control with motor encoder
0x00	A	0103	Position control with external encoder
0x00	A	0104	Position control with motor encoder and lagless function
0x00	A	0105	Position control with external encoder and lagless function
0x00	A	0106	Drive controlled interpolation with motor encoder
0x00	A	0107	Drive controlled interpolation with external encoder
0x00	A	0108	Drive controlled interpolation with motor encoder and lagless function
0x00	A	0109	Drive controlled interpolation with external encoder and lagless function
0x00	A	0150	Drive controlled positioning with motor encoder
0x00	A	0151	Drive controlled positioning with motor encoder and lagless function
0x00	A	0152	Drive controlled positioning with external encoder
0x00	A	0153	Drive-controlled positioning with external encoder and lagless function
0x00	A	0158	Spool control
0x00	A	0170	Pressure control
0x00	A	0171	Flow control
0x00	A	0172	Pressure/flow control
0x00	A	4000	Automatic drive check and adjustment
0x00	A	4001	Drive deceleration to standstill

7.8.2 Drive Status code “procedure command specific state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x00	C	300	Set absolute position procedure command
0x00	C	301	Measuring system unavailable
0x00	C	302	Absolute evaluation of measuring system impossible
0x00	C	303	reserved
0x00	C	304	Set absolute position cannot be executed
0x00	C	600	Drive-controlled homing procedure command
0x00	C	601	Homing possible with drive enable only
0x00	C	602	Distance from home switch to reference marker is erroneous
0x00	C	603	Homing impossible with external encoder
0x00	C	604	reserved
0x00	C	605	reserved
0x00	C	606	Reference mark not detected
0x00	C	900	Position spindle procedure command
0x00	C	902	Spindle positioning requires drive enable
0x00	C	906	Error during search for marker pulse
0x00	C	1300	Positive stop drive procedure command
0x00	C	1301	Positive stop is activated and C1D error is present
0x00	C	1500	Cancel reference point procedure command
0x00	C	1600	Parking axis procedure command
0x00	C	3300	Set coordinate system procedure command
0x00	C	3400	Shift coordinate system procedure command
0x00	C	4100	Switch parameter set procedure command
0x00	C	4101	Switching possible only without drive enable
0x00	C	4102	reserved
0x00	C	4103	Preselect parameter set forbidden value
0x00	C	4104	Error during parameter set switching
0x00	C	4200	Gear engaging procedure command
0x00	C	4201	Gear engaging requires drive enable
0x00	C	4202	Gear engaging speed cannot be reached
0x00	C	4300	NC-controlled homing procedure command
0x00	C	4302	Distance from home switch to reference mark is erroneous
0x00	C	4306	Reference mark not detected
0x00	C	4307	Home switch not assigned
0x00	C	4308	Homing of modulo axes is not allowed
0x00	C	4400	Calculate displacement procedure command
0x00	C	4500	Displacement to referenced system procedure command

7.8.3 IO Status codes of “operational state”, “warning” and “error”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x01	A or E or F	0	No error
0x01	A or E or F	1000	General error
0x01	A or E or F	2000	Current
0x01	A or E or F	2100	Current on the device input side
0x01	A or E or F	2110	Short circuit/ground fault
0x01	A or E or F	2120	Ground fault
0x01	A or E or F	2130	Short circuit
0x01	A or E or F	2136	Short circuit to VCC
0x01	A or E or F	2137	Short circuit to Ground
0x01	A or E or F	2200	Current inside the device
0x01	A or E or F	2300	Current on the device output side
0x01	A or E or F	2310	Continuous over current
0x01	A or E or F	2320	Short circuit/ground fault
0x01	A or E or F	2330	Ground fault
0x01	A or E or F	2340	Short circuit
0x01	A or E or F	2344	Output overload
0x01	A or E or F	2345	Sensor supply overload
0x01	A or E or F	2346	Short circuit to VCC
0x01	A or E or F	2347	Short circuit to Ground
0x01	A or E or F	2350	Open circuit
0x01	A or E or F	2360	Wire break
0x01	A or E or F	3000	Voltage
0x01	A or E or F	3100	Mains voltage (Voltage on the device input side -power supply)
0x01	A or E or F	3110	Mains surge voltage
0x01	A or E or F	3120	Mains undervoltage
0x01	A or E or F	3134	Phase sequence
0x01	A or E or F	3140	Mains frequency
0x01	A or E or F	3200	Voltage inside the device
0x01	A or E or F	3210	Surge voltage inside the device
0x01	A or E or F	3220	Undervoltage inside the device
0x01	A or E or F	3300	Output voltage (Voltage on the device output side e.g. supply or working voltage for external parts)
0x01	A or E or F	3310	Output surge voltage
0x01	A or E or F	3320	Output undervoltage
0x01	A or E or F	3400	Supply voltage
0x01	A or E or F	3410	Sensor supply
0x01	A or E or F	3420	Actuator supply
0x01	A or E or F	4000	Temperature
0x01	A or E or F	4100	Ambient temperature
0x01	A or E or F	4200	Device temperature
0x01	A or E or F	4300	External temperature (e.g. drive)
0x01	A or E or F	4400	Supply temperature
0x01	A or E or F	5000	Device hardware (only inside the device housing)
0x01	A or E or F	5010	Component error

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x01	A or E or F	5100	Supply
0x01	A or E or F	5110	Supply low voltage (for all internal voltages 24V, 5V, 3.3V, 3.0V, 2.5V, 1.8V)
0x01	A or E or F	5120	Supply for air
0x01	A or E or F	5150	Supply for initiator (Error of the supply for an external sensor supplied from inside the device)
0x01	A or E or F	5160	Supply for peripheral devices (Error of the supply for external peripherals supplied from inside the device)
0x01	A or E or F	5200	Control system
0x01	A or E or F	5210	Measuring circuits
0x01	A or E or F	5220	Computer circuits
0x01	A or E or F	5230	Communication (inside the device)
0x01	A or E or F	5300	Operating and display unit
0x01	A or E or F	5400	Power section
0x01	A or E or F	5410	Output stages
0x01	A or E or F	5420	Choppers
0x01	A or E or F	5430	Input stages
0x01	A or E or F	5440	Contactors
0x01	A or E or F	5450	Fuses
0x01	A or E or F	5500	Communication with add-on module
0x01	A or E or F	6000	Device software
0x01	A or E or F	6010	Software reset *(watchdog)*
0x01	A or E or F	6100	Internal software *(firmware)*
0x01	A or E or F	6200	Application software
0x01	A or E or F	6300	Data record not OK
0x01	A or E or F	6310	Parameter missing
0x01	A or E or F	6320	Parameter error
0x01	A or E or F	6330	Parameter not yet initialized
0x01	A or E or F	7000	Add-on module(s) faulty
0x01	A or E or F	7100	Power
0x01	A or E or F	7200	Measuring circuit
0x01	A or E or F	7300	Sensor
0x01	A or E or F	7400	Computer circuit
0x01	A or E or F	7500	Communication
0x01	A or E or F	7600	Data memory
0x01	A or E or F	7700	Open circuitcable error
0x01	A or E or F	8000	Monitoring
0x01	A or E or F	8100	Communication
0x01	A or E or F	8110	Process data monitoring
0x01	A or E or F	8120	Host monitoring
0x01	A or E or F	8200	Closed-loop control
0x01	A or E or F	8210	System deviation high (desired actual, manufacturer specific) - (deviation present longer than a specified period of time)
0x01	A or E or F	8211	Maximum manipulated variable reached
0x01	A or E or F	8220	System deviation (desired actual, manufacturer specific) - (deviation present longer than a specified period of time)
0x01	A or E or F	8221	Maximum manipulated variable reached

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x01	A or E or F	8900	Sensors
0x01	A or E or F	8910	Measured value overrange
0x01	A or E or F	8920	Measured value underrange
0x01	A or E or F	8A00	Actuators
0x01	A or E or F	8B00	Preventive maintenance required (condition monitoring)
0x01	A or E or F	9000	External error
0x01	A or E or F	A000	Sub bus slave
0x01	A or E or F	A001	Sub bus slave missing
0x01	A or E or F	A002	Incorrect Sub bus slave present
0x01	A or E or F	A003	Compatible replacement (the expected Sub bus slave was replaced by a compatible one)
0x01	A or E or F	A004	Sub bus devices amount error (too many devices)
0x01	A or E or F	A010	Device Errors
0x01	A or E or F	A012	Application on the Sub bus slave not ready
0x01	A or E or F	A013	Sub bus device reset
0x01	A or E or F	A020	Communication errors
0x01	A or E or F	A021	Sub bus error - timeout (Break)
0x01	A or E or F	A022	Multiple transmission error on Sub bus
0x01	A or E or F	A023	Sub bus I/O data communication error
0x01	A or E or F	A024	Sub bus management data communication error
0x01	A or E or F	A030	Sub bus configuration error
0x01	A or E or F	A031	I/O data configuration double mapped
0x01	A or E or F	A040	Common Errors
0x01	A or E or F	A041	Sub bus hardware error
0x01	A or E or F	A042	Sub bus firmware error
0x01	A or E or F	A043	Sub bus is running asynchronous (in relation to the Sercos bus cycle)
0x01	A or E or F	B000	Local bus (Systembus of the modular station)
0x01	A or E or F	B001	Local bus slave missing
0x01	A or E or F	B002	Incorrect local bus slave present
0x01	A or E or F	B003	Compatible replacement (the expected local bus slave was replaced by a compatible one)
0x01	A or E or F	B004	Local bus devices amount error (too much devices)
0x01	A or E or F	B010	Device Errors
0x01	A or E or F	B012	Application on the local bus slave not ready
0x01	A or E or F	B013	Local bus device reset
0x01	A or E or F	B020	Communication errors
0x01	A or E or F	B021	Local bus error - timeout (Break)
0x01	A or E or F	B022	Multiple transmission error on local bus
0x01	A or E or F	B023	Local bus I/O data communication error
0x01	A or E or F	B024	Local bus management data communication error
0x01	A or E or F	B030	Local bus configuration error
0x01	A or E or F	B031	I/O data configuration double mapped
0x01	A or E or F	B040	Common Errors
0x01	A or E or F	B041	Local bus hardware error
0x01	A or E or F	B042	Local bus firmware error
0x01	A or E or F	B043	Local bus is running asynchronous (in relation to the Sercos bus cycle)
0x01	A or E or F	F000	Additional functions

7.8.4 IO Status codes “procedure command specific state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x01	C	BC00	S-0-1500.x.12 Rearrangement of IO resource
0x01	C	BC01	Forbidden insertion/replacement - An attempt was made to insert/replace a generic IO function group with an invalid/forbidden slot number
0x01	C	BC02	Forbidden insertion/replacement - An attempt was made to insert/replace the function group IO bus coupler (S-0-1500.x.0)
0x01	C	BC03	Forbidden insertion/replacement - An attempt was made to insert/replace an undefined generic IO function group
0x01	C	BC04	Forbidden insertion/replacement - The list of inserted/replaced function groups (S-0-1500.x.4/11) contains duplicate entries
0x01	C	BC05	Forbidden insertion/replacement - The list of inserted/replaced function groups (S-0-1500.x.4/11) is not sorted in an increasing order (slot number, data block number)
0x01	C	BC0F	Forbidden insertion/replacement - Manufacturer-specific reason

7.8.5 GDP Status codes “operational state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x02	A	A010	The device has been restarted (Power on)
0x02	A	A100	Wrong password entered
0x02	A	A110	Password write protection deactivated
0x02	A	A120	Password changed
0x02	A	A200	Diagnostic trace started
0x02	A	A210	Diagnostic trace stopped
0x02	A	A220	Diagnostic trace buffer overrun
0x02	A	A300	Test IDN written

7.8.6 GDP Status codes “procedure command specific state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x02	C	0200	S-0-0422 Exit parameterization level procedure command
0x02	C	0201	Incorrect or incomplete set of parameters (see S-0-0423 IDN-list of invalid data for parameterization level)
0x02	C	0202	Parameter limit violation (see S-0-0423 IDN-list of invalid data for parameterization level)
0x02	C	0203	Parameter conversion error (see S-0-0423 IDN-list of invalid data for parameterization level)
0x02	C	0400	S-0-0420 Activate parametrization level procedure command (PL)
0x02	C	0401	Switching to parametrization level is not possible
0x02	C	0500	S-0-0099 Reset class 1 diagnostic
0x02	C	0700	S-0-0262 Load defaults procedure command
0x02	C	2200	S-0-0264 Backup working memory procedure command
0x02	C	2300	S-0-0263 Load working memory procedure command
0x02	C	2400	S-0-0293 Selectively backup working memory procedure command

7.8.7 FSP Status codes “operational state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x03	A	0000	Communication phase 0
0x03	A	0001	Communication phase 1
0x03	A	0002	Communication phase 2
0x03	A	0003	Communication phase 3
0x03	A	0004	this status code shall not be used
0x03	A	0005	this status code shall not be used
0x03	A	0006	this status code shall not be used
0x03	A	0007	this status code shall not be used
0x03	A	0008	NRT state
0x03	A	0009	this status code shall not be used
0x03	A	0030	Hot-plug phase 0
0x03	A	0031	Hot-plug phase 1
0x03	A	0032	Hot-plug phase 2

7.8.8 FSP Status codes “procedure command specific state”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x03	C	100	S-0-0127 CP3 transition check
0x03	C	101	Invalid parameters
0x03	C	104	Configured IDNs for MDT not configurable
0x03	C	105	Maximum length for MDT exceeded
0x03	C	106	Configured IDNs for AT not configurable
0x03	C	107	Maximum length for AT exceeded
0x03	C	108	Timing parameter > Sercos cycle time (tScyc)
0x03	C	109	Telegram offset unsuitable
0x03	C	110	this status code shall not be used
0x03	C	111	this status code shall not be used
0x03	C	112	this status code shall not be used
0x03	C	113	this status code shall not be used
0x03	C	114	this status code shall not be used
0x03	C	115	this status code shall not be used
0x03	C	116	this status code shall not be used
0x03	C	139	this status code shall not be used
0x03	C	170	Configured IDNs for connection not configurable
0x03	C	171	Maximum length for connections exceeded
0x03	C	172	S-0-1024 SYNC delay measuring procedure command not performed
0x03	C	173	Quantity of connections is not configurable
0x03	C	174	Connection configuration is not possible
0x03	C		
0x03	C	175	Producer cycle time (tPcyc) of a connection is wrong
0x03	C	176	SCP classes not correct configured
0x03	C	177	Timing parameter are changed, see S-0-0021 IDN-list of invalid operation data for CP2
0x03	C	5200	S-0-0128 CP4 transition check
0x03	C	5300	S-0-1024 SYNC delay measuring procedure command
0x03	C	5301	S-0-1024 SYNC delay measuring procedure command failed
0x03	C	5302	S-0-1024 SYNC delay measuring procedure command error

7.8.9 FSP Status codes “warning”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x03	E	4001	Warning of MST losses
0x03	E	4002	RTD-failure shutdown
0x03	E	4007	Consumer connection failed
0x03	E	4008	Invalid addressing of MDT data container A
0x03	E	4009	Invalid addressing of AT data container A
0x03	E	4010	reserved
0x03	E	4019	CPS=1 and master changes the CP to an invalid value.
0x03	E	4020	Topology status changes from fast-forward (FF) to loop-back with forward (L&F)
0x03	E	4030	OVS producer data are invalid
0x03	E	4031	OVS container length exceeded
0x03	E	4032	OVS sample factor not allowed or not supported
0x03	E	4033	OVS configuration is incorrect
0x03	E	4034	OVS configuration doesn't match
0x03	E	4035	reserved for OVS

7.8.10 FSP Status codes “error”

Bit 29-24 source type	Bit 19-16 class (hex)	Bit 15-0 code (hex)	Description
0x03	F	4001	Error of MST losses
0x03	F	4002	connection losses
0x03	F	4017	CPS-MST timeout (500ms) occurs during phase switch
0x03	F	4019	CPS=0 and master changes the CP to an invalid value.
0x03	F	4020	Topology status changes from Loop-back with Forward to NRT mode
0x03	F	4021	Slave doesn't support the announced Communication Version for CP1 and CP2

8 Appendix

8.1 Protecting IDNs with a Password

This chapter describes how to protect IDNs with a password. The necessary Sercos IDNs:

- S-0-0279 IDN list of password protected data
- S-0-0267 Password

are still created in the stack. Also the handling for checking the password if a protected IDN shall get accessed is included in the stack. But to activate this behaviour a list of IDNs to protect must be provided to the stack. Also a hash of a password must be provided. The following figure explains the configuration and the needed packets to activate password protection.

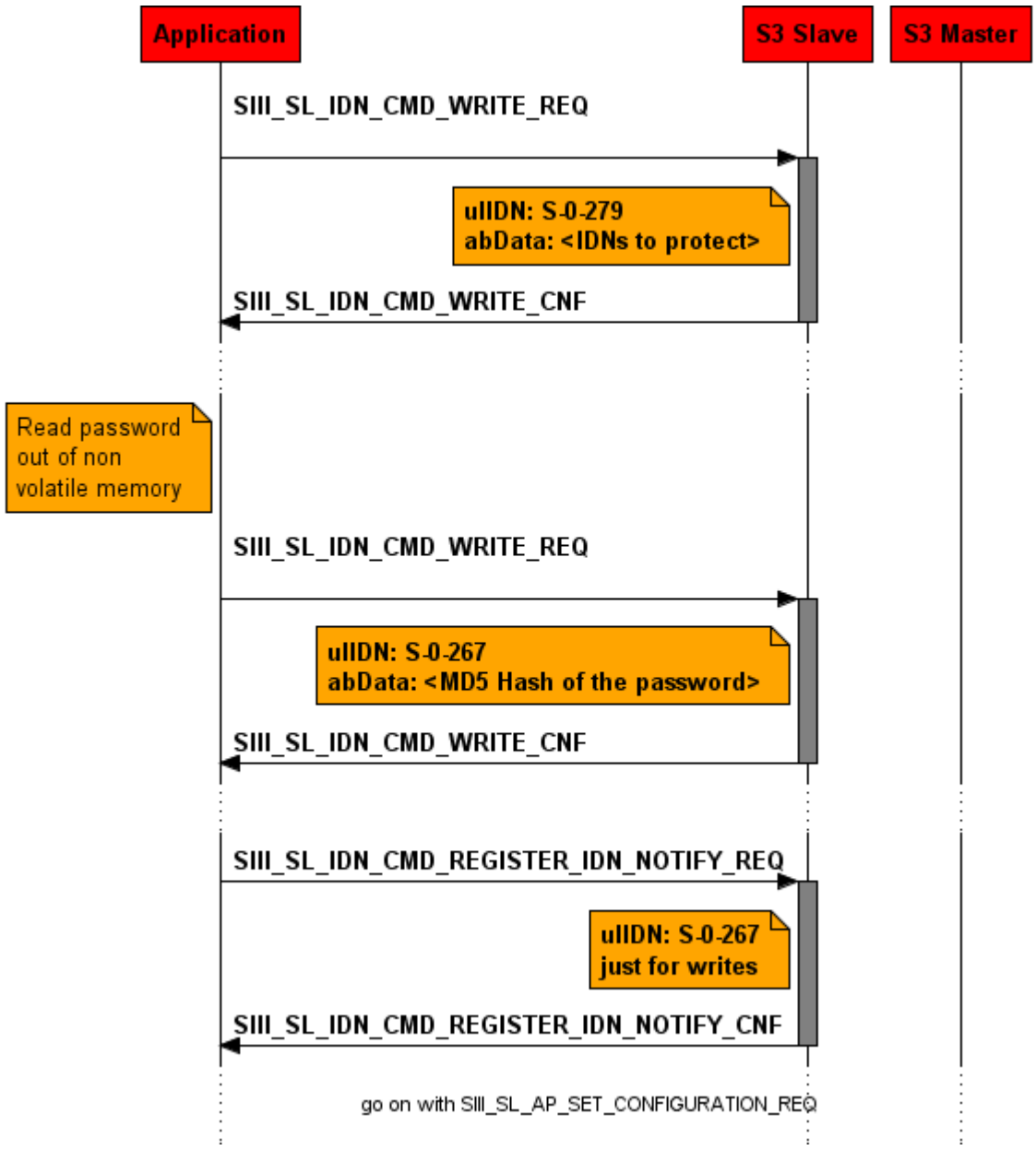


Figure 78: Configure password protected IDNs

It is necessary to register for writes on S-0-0267 (Password). The stack will just send a write indication if the master has changed the password. In this case the new password-hash must be

stored non-volatile. Use the stored password hash at startup after a power cycle for stack configuration.

Nothing else must be done. The stack will check the password if a protected IDN gets accessed.

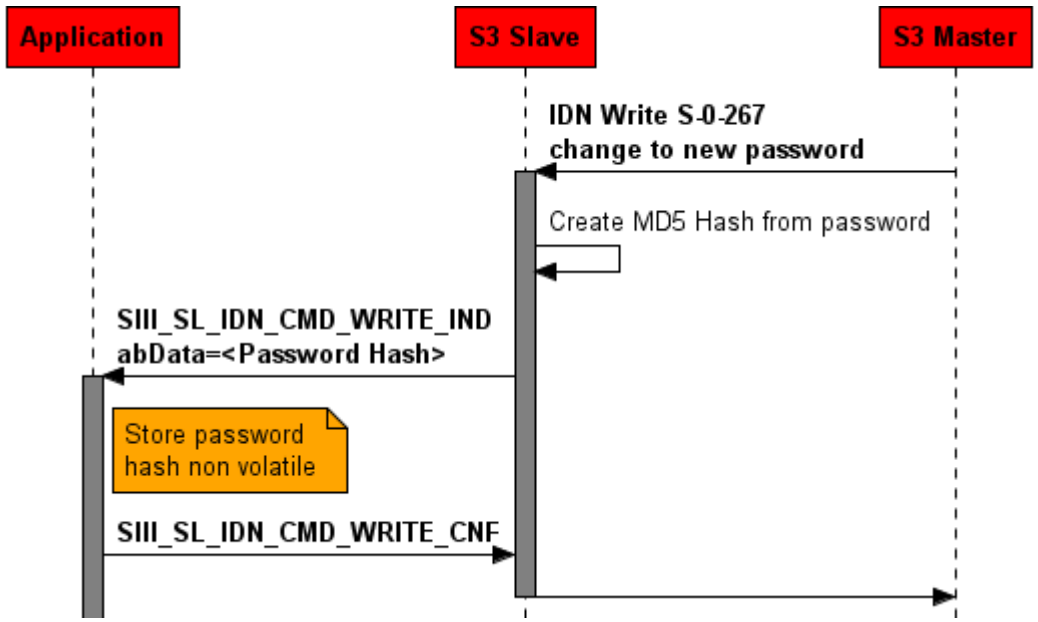


Figure 79: Change the password in the non volatile storage

8.2 IDNs created by the Stack

The following IDNs are created by the stack. It is not allowed to create one of the IDNs by the application. Otherwise, an error will be reported.

IDN	Task	Description
S-0-0014	COM-Task	Interface status
S-0-0015	COM-Task	Telegram Type
S-0-0017	IDN-Task	IDN list of all operation data
S-0-0021	COM-Task	IDN-list of invalid operation data for CP2
S-0-0022	COM-Task	IDN-list of invalid operation data for CP3
S-0-0025	IDN-Task	IDN list of all procedure commands
S-0-0095	COM-Task	Diagnostic message
S-0-0099	COM-Task	Reset class 1 diagnostic
S-0-0127	COM-Task	CP3 transition check
S-0-0128	COM-Task	CP4 transition check
S-0-0279	IDN-Task	IDN list of password protected data
S-0-0267	IDN-Task	Password
S-0-0390	COM-Task	Diagnostic number
S-0-1000	COM-Task	SCP Type & Version
S-0-1002	COM-Task	Communication cycle time (tScyc)
S-0-1003	COM-Task	Allowed MST losses in CP3 & CP4
S-0-1005	COM-Task	Minimum feedback processing time (t5)
S-0-1006	COM-Task	AT0 transmission starting time (t1)
S-0-1007	COM-Task	Synchronization time (TSync)
S-0-1008	COM-Task	Command value valid time (t3)
S-0-1009	COM-Task	Device Control (C-Dev) offset in MDT
S-0-1010	COM-Task	Length of MDTs
S-0-1011	COM-Task	Device Status (S-Dev) offset in AT
S-0-1012	COM-Task	Length of ATs
S-0-1013	COM-Task	SVC offset in MDT
S-0-1014	COM-Task	SVC offset in AT
S-0-1019	NRT-Task	MAC Address
S-0-1015	COM-Task	Ring delay
S-0-1016	COM-Task	Slave delay (P&S)
S-0-1017	COM-Task	NRT transmission time
S-0-1020	NRT-Task	IP Address
S-0-1020.0.1	NRT-Task	Current IP address
S-0-1021	NRT-Task	Subnet Mask
S-0-1021.0.1	NRT-Task	Current active subnet mask
S-0-1022	NRT-Task	Gateway address
S-0-1022.0.1	NRT-Task	Current active gateway address
S-0-1023	COM-Task	SYNC jitter
S-0-1024	COM-Task	AT command value valid time (t9)
S-0-1026	COM-Task	Version of communication hardware
S-0-1027.0.1	NRT-Task	Requested MTU
S-0-1027.0.2	NRT-Task	Effective MTU
S-0-1028	COM-Task	Error Counter MST-P&S

IDN	Task	Description
S-0-1032	COM-Task	Communication control
S-0-1035	COM-Task	Error counter Port 1 & Port 2
S-0-1035.0.1	COM-Task	Error counter P&S
S-0-1036	COM-Task	Inter frame Gap
S-0-1037	COM-Task	Slave Jitter
S-0-1039	NRT-Task	Hostname
S-0-1039.0.1	NRT-Task	Current active hostname
S-0-1040	COM-Task	Sercos address
S-0-1044	COM-Task	Device Control
S-0-1045	COM-Task	Device Status
S-0-1046	COM-Task	List of sercos address in device
S-0-1048	NRT-Task	Activate network settings
S-0-1051	COM-Task	Image of connection setups
S-0-1300.0.3	COM-Task	Vendor Code
S-0-1300.0.4	COM-Task	Vendor Device ID
S-0-1300.0.8	AP-Task	Hardware Revision (IDN created at LFW only)
S-0-1300.0.9	AP-Task	Software Revision (IDN created at LFW only)
S-0-1301	COM-Task	List of GDP Classes & Version
S-0-1302.0.1	COM-Task	FSP Type & Version
S-0-1303.0.2	COM-Task	Diagnostic trace control
S-0-1303.0.3	COM-Task	Diagnostic Trace State
S-0-1303.0.10	COM-Task	Diagnostic trace buffer no1
S-0-1303.0.11	COM-Task	Diagnostic trace buffer no2
S-0-1305.0.1	COM-Task	Sercos current time
S-0-1399	COM-Task	Diagnostic Event

Table 114: Power-on IDNs

IDNs with an SI = x are created for each configured instance.

If in the slave receives the packet SIII_SL_AP_CMD_SET_CONFIGURATION_REQ and the flag usSlaveFlags is set to SIII_SL_AP_SET_CONFIGURATION_SLAVE_FLAGS_SETUP_DEFAULT_OD the slave will create the following IDNs after a channel init:

IDN	Task	Description
S-0-1050.x.8	COM-Task	Connection Control (C-Con)
S-0-1302.0.2	COM-Task	Function groups
S-0-1500.0.1	COM-Task	IO Control
S-0-1500.0.2	COM-Task	IO Status
S-0-1500.0.3	COM-Task	List of module type codes
S-0-1500.0.5	COM-Task	Container OutputData
S-0-1500.0.9	COM-Task	Container InputData
S-0-1502.0.3	COM-Task	Channel Quantity PDOUT
S-0-1502.0.4	COM-Task	Channel Width PDOUT
S-0-1503.0.7	COM-Task	Channel Quantity PDIN
S-0-1503.0.8	COM-Task	Channel Width PDIN

Table 115: Optional IDNs

IDNs with an SI = x are created for each configured instance.

If in the slave receives the packet SIII_SL_AP_CMD_SET_CONFIGURATION_REQ the slave will create the following IDNs for each configured connection after a channel init:

IDN	Task	Description
S-0-1050.x.1	COM-Task	Connection setup
S-0-1050.x.2	COM-Task	Connection Number
S-0-1050.x.3	COM-Task	Telegram Assignment
S-0-1050.x.4	COM-Task	Max. Length of Connection
S-0-1050.x.5	COM-Task	Current Length of connection
S-0-1050.x.6	COM-Task	Configuration List
S-0-1050.x.10	COM-Task	Producer Cycle Time
S-0-1050.x.11	COM-Task	Allowed Data Losses
S-0-1050.x.12	COM-Task	Error Counter Data Losses

Table 116: IDNs depending on the number of used connections

IDNs with an SI = x are created for each configured instance.

NRTPC V2

If bSCP_NRT_Version is set to 2 (`SIII_SL_AP_CMD_SET_CONFIGURATION_REQ` service) the stack created the following IDNs:

IDN	Task	Description
S-0-1048.0.1	NRT-Task	Configuration of IP options
S-0-1048.0.2	NRT-Task	Current IP options
S-0-1049.0.0	NRT-Task	List of IPS classes and version

Table 117: IDNs for NRTPC V2

8.3 Sync-clock and Div-clock Interrupts

This section describes how the user application can access to the sync and divided clock interrupts. To use these, the user application has to run on the netX chip together with the Sercos slave stack (LOM).

Note: The callback must be registered before entering CP3. It is recommended to register the callback after system startup.

If a callback is registered, at every sync/div-clock interrupt the registered function will be called. The sync function will be called once per Sercos cycle. The div-clock interrupt is called depending on the configuration.

The following Interrupts can be registered:

```
typedef enum S3SLAVE_INTERRUPT_SOURCE_Ttag
{
  S3SLAVE_INTERRUPT_SOURCE_CON_CLK = 2,
  S3SLAVE_INTERRUPT_SOURCE_DIV_CLK = 3,
} S3SLAVE_INTERRUPT_SOURCE_T;
```

Use the following example code to register for a con-clock or div-clock interrupt. It is necessary to provide a pointer to an own function and a pointer to local resources.

The calling function needs an interface handle. This will be returned by the HAL function S3s_GetInterface().

```
{
/* register at task startup */

S3SLAVE_INTERFACE_HANDLE            hInterface;
hInterface = S3s_GetInterface(); /* get handle of Hal task */

/* Register the Sync Interrupt */
S3Slave_SetInterrupt(hInterface,
                     S3SLAVE_INTERRUPT_SOURCE_CON_CLK,
                     Function_Called_Sync,
                     ptUserData);

/* Register the Sync Interrupt */
S3Slave_SetInterrupt(hInterface,
                     S3SLAVE_INTERRUPT_SOURCE_DIV_CLK,
                     Function_Called_Sync,
                     ptUserData);
}
```

Example of an application function which shall be called by a con-clock or div-clock interrupt.

```
Void Function_Called_Sync( S3SLAVE_INTERRUPT_SOURCE_T  eInterrupt,
void* pvUserData)
{
  /* User Function */
  switch(eInterrupt)
  {
     case S3SLAVE_INTERRUPT_SOURCE_CON_CLK:
     /* con-clock */
     break;

     case S3SLAVE_INTERRUPT_SOURCE_DIV_CLK:
     /* div-clock */
     break;
  }
}
```

8.4 Migration from Stack Version V3.0 to V3.1

Take care of the two following topics when migrating from Sercos stack V3.0 to V3.1

8.4.1 Update of `config.c` File

Note: The update of the `config.c` file described here is **only necessary** when working with the **linkable objects version (LOM)** of the Sercos Slave protocol stack V3.1.

Two adaptations must be made in the `config.c` file. Otherwise a compiler error will occur.

These adaptations are:

1. Correct `ausSercosAddresses`

   Search for the following structure within the `config.c` file.

```
static const S3S_CONFIG_SET_T g_tS3sConfig =
{
  /* timer number used for NRT watchdog */
  .uiNrtWdgTimerNo = 1,
  .uiNumSlavesInDevice = 1,
  .ausSercosAddresses =
  {
    1,
    2,
    3,
    4,
    5,
    6,
    7,
    8
  }
};
```

At the second parameter, you should find the following:

```
  /* timer number used for measuring the cycle length in CP0-CP2 */
  .uiCycleMeasurementTimerNo = 2,
```

   Remove this assignment.

2. The second adaptation only affects the netX 100 and netX 500 communication controller: The define statement of the Sync Interrupt has changed.

   In order to correct it, just replace `define SRT_NETX_VIC_IRQ_STAT_com3` by `define SRT_NETX_VIC_IRQ_STAT_msync3`.

8.4.2 Change in Sync Configuration

The clock parameters `ulDTDivClk` (Contents of DTDivClk register) and `ulDivClkLength` (DivClk Length) are no longer evaluated (since V3.1). `ulDivClkLength` is now always set to the fixed value 1 µs. In Div_Clk mode 0 it is not possible any more to configure aperiodic signals. ulConClkLength is now limited to a maximum of 655350 ns.

8.5 Migration from Stack Version V3.1 to V3.2

Take care of the following topic when migrating from Sercos stack V3.1.x.x to V3.2.x.x.

With Stack V3.2 the new feature hot-plug is build in. If the slave performs hot-plug the new phases will be reported by the communication phase change indication. If your device executes own code at Phase CP2 or CP3 entry this must also be done for the HP phases. Please review your code while receiving CP2 and CP3 phase change indications and extend the handling if necessary.

For netX51 and netX52 Targets remove the “S3S_NRTWDG” from the definition at RX_INTERRUPT_SET_T. On netX51 and netX52 change RX_HWTIMER_SET_T to

```
{
  {"S3S_NRTWDG",RX_PERIPHERAL_TYPE_INTERRUPT,0},
  SRT_NX51_vic_irq_status_gpio_timer, /* changed */
  27, /* Target specific */
  RX_INTERRUPT_MODE_SYSTEM,
  RX_INTERRUPT_EOI_AUTO,
  RX_INTERRUPT_TRIGGER_LEVEL_ONE, /* changed */
  RX_INTERRUPT_PRIORITY_STANDARD,
  RX_INTERRUPT_REENTRANCY_DISABLED,
},
```

8.6 Migration from Stack Version V3.2 to V3.3

Stack V3.3 provides the option to configure more connections. If a slave shall use more than four conections, this can be done with the new packet SIII_SL_COM_CMD_CHANGE_CONNECTION_DATA_OFFSETS_REQ.

A registration for all IDNs listed in Table 115 on page 227 and Table 116 on page 227 cannot be done prior to Channel Init.

8.7 Migration from Stack Version V3.3 to V3.4

No special issues need to be taken into account when migrating from stack version V3.3 to V3.4.

8.8 Migration from Stack Version V3.4 to V3.5

Parameter bSercosVersion (Table 19 on page 63) was a reserved parameter before: value 0 activates that the stack runs compatible to Sercos Version 1.1.2. If required that the stack runs compatible to Sercos Version 1.3.1, the application has to set parameter bSercosVersion to 1.

8.9 List of Tables

8.10 List of Figures

8.11 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